

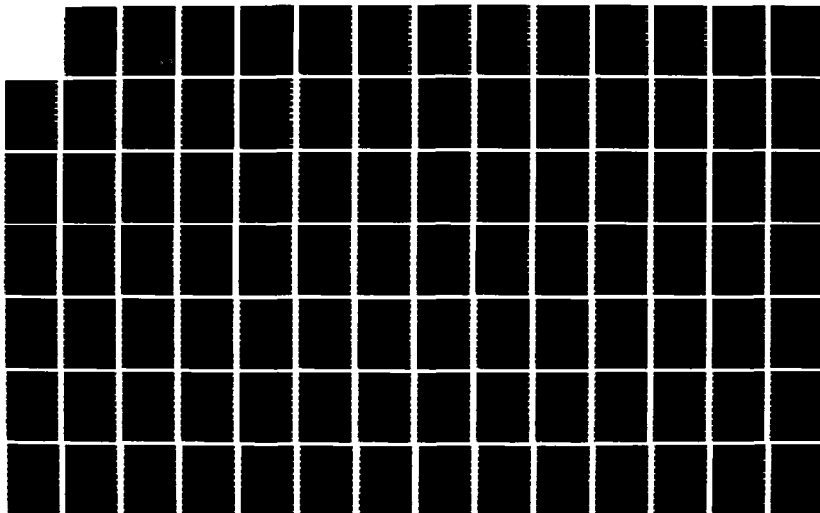
BD-A174 450

COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION USING A  
DATABASE MANAGEMENT (U) IOWA UNIV IOWA CITY OPTIMAL  
DESIGN LAB T SREEKANTAMURTHY ET AL 30 SEP 86  
ODL-85-17 AFOSR-TR-86-2069 AFFSR-82-0322 F/G 18/3

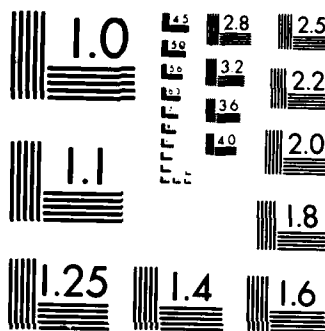
1/4

UNCLASSIFIED

NL







MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A



AD-A174 450

AFOSR-TR- 86 - 2069

Technical Report No. ODL-85.17

# Computer-Aided Structural Design Optimization Using A Database Management System

T. Sreekanta Murthy and J. S. Arora

Optimal Design Laboratory

College of Engineering  
The University of Iowa  
Iowa City, Iowa 52242

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)  
NOTICE OF TECHNICAL REPORT (AFSC)  
THIS TECHNICAL REPORT HAS BEEN APPROVED AND IS  
APPROVED FOR PUBLICATION BY AFSC AFOSR-85-12.  
DISTRIBUTION IS UNLIMITED.  
MATTHEW J. HERTZ  
Chief, Technical Information Division

Prepared for the  
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH  
Under Grant No. AFOSR-82-0322

September 1985

DISTRIBUTION STATEMENT A  
Approved for public release  
Distribution Unlimited

DTIC  
ELECTE  
NOV 26 1986  
S D

86 11 25 293



ADA174450

## REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION AVAILABILITY OF REPORT  Approved for unlimited distribution	
2b. DECLASSIFICATION DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ODL-85.17		5. MONITORING ORGANIZATION REPORT NUMBER(S) <b>AFOSR-TR- 86 - 2069</b>	
6a. NAME OF PERFORMING ORGANIZATION Optimal Design Laboratory	6b. OFFICE SYMBOL (If applicable) ODL	7a. NAME OF MONITORING ORGANIZATION AFOSR/NA	
6c. ADDRESS (City, State and ZIP Code) College of Engineering The University of Iowa Iowa City, IA 52242		7b. ADDRESS (City, State and ZIP Code) Bolling AFB D.C. 20332-6448	
8a. NAME OF FUNDING SPONSORING ORGANIZATION Air Force Office of Scientific Research	8b. OFFICE SYMBOL (If applicable) NA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER  Grant No. AFOSR-82-0322	
8c. ADDRESS (City, State and ZIP Code) 3D 410 Aerospace Sciences Bolling AFB, DC 20332-6448		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) Computer-Aided Structural Design Optimization Using a DBMS		PROGRAM ELEMENT NO. 61102	PROJECT NO. 2307
12. PERSONAL AUTHOR(S) T. FreekantaMurthy and J.S. Arora		TASK NO. B1	WORK UNIT NO.
12a. TYPE OF REPORT Interim	12b. TIME COVERED FROM 10/84 TO 9/85	14. DATE OF REPORT (Yr - Mo - Day) 1985-9-30	15. PAGE COUNT 319
13. SUPPLEMENTARY NOTES			
16. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Design Optimization, Database Design, Database Management System, Evaluation, Structures.		17. ABSTRACT (Continue on reverse if necessary and identify by block number) A methodology to integrate finite element-based-optimal structural design methods and computer-aided methods into a computer-based system containing a database, a program library and man-machine communication link. Emphasis is placed upon database management concepts for structural design. Important components required to build a computer-aided structural design system are described. A number of database management concepts -- hierarchical, network and relational data models, conceptual, internal and external view of data organization, normalization of data, and global and local database are discussed with reference to structural design data. A methodology to design a database is proposed. Three levels of data organization--conceptual, internal and external are suggested. A methodology to construct a numerical data model is described. This model supports data of various types of structures such as banded, skyline and hypermatrices. Requirements of database management system and components needed to develop it are discussed. Language requirements for the computer-aided communication link between designer and computer are formulated. A database management system (DBMS) is implemented for use in structural design applications. MIMAS	
18. ABSTRACT SECURITY CLASSIFICATION		19. ABSTRACT SECURITY CLASSIFICATION	
20. NAME OF PERFORMING ORGANIZATION AFOSR/NA		21. TELEPHONE NUMBER (Include area code) 202-267-4617	
22. NAME OF PERFORMING ORGANIZATION AFOSR/NA		23. OFFICE SYMBOL NA	



supports both relational and numerical data models. It can be used either through application program calls or interactively. A database for structural design optimization is designed using the proposed methodology. A computer program for finite element analysis and structural design optimization is developed. The program uses database and the database management system MIDAS. The program is based on hypermatrix approach for assembly and solution of large matrix equations. Several example problems are solved using the program. An evaluation of data model, database and database management system in computer-aided structural design optimization environment is made. The performance of MIDAS in equation solving environment is determined using skyline and hypermatrix approach. Finally, it is concluded that with the proposed data models, database design methodology, and the advanced database management system computer-aided design optimization of complex structural system can be attempted.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE



Technical Report No. ODL-85.17

**COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION  
USING A DATABASE MANAGEMENT SYSTEM**

T. Sreekanta Murthy and J.S. Arora

**Optimal Design Laboratory**  
College of Engineering  
The University of Iowa  
Iowa City, Iowa 52242

Prepared for the  
AIRFORCE OFFICE OF SCIENTIFIC RESEARCH  
Under Grant No. AFOSR-82-0322

September 1985



## PREFACE

This is one of a series of reports under the Grant No. AFOSR 82-0322 from the Air Force Office of Scientific Research, Air Force Systems Command, USAF. The project title is "Database Design and Management in Engineering Design Optimization." The report is based on the research conducted between October 1982 to September 1985 leading to the Ph.D. dissertation of Mr. T. SreekantaMurthy completed under the direction of Professor Jasbir S. Arora.



Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification:	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



**COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION  
USING A DATABASE MANAGEMENT SYSTEM**

by

Thammaiah Creekantamurthy

An Abstract

Of a thesis submitted in partial fulfillment of  
the requirements for the Doctor of Philosophy  
degree in Civil and Environmental Engineering  
in the Graduate College of  
The University of Iowa

December 1985

Thesis supervisor: Professor Jasbir S. Arora



### ABSTRACT

A study <sup>was</sup> ~~is~~ made to integrate finite element-based-optimal structural design methods and computer-science methods into a computer-based system containing a database, a program library and man-machine communication link. Emphasis is placed upon database management concepts for structural design. Important components required to build a computer-aided structural design system are described. A number of database management concepts -- hierarchical, network and relational data models, conceptual, internal and external view of data organization, normalization of data, and global and local database are discussed with reference to structural design data. A methodology to design a database is proposed. Three levels of data organization--conceptual, internal and external are suggested. A methodology to construct a numerical data model is described. This model supports data of various types of large matrices such as banded, skyline and hypermatrices. Requirements of database management system and components needed to develop it are discussed. Language requirements to enable good communication link between designer and computer are formulated. A database management system - MIDAS is implemented for use in structural design applications. MIDAS supports both relational and numerical data models. It can be used either through application program calls or interactively. A database for structural design



optimization is designed using the proposed methodology. A computer program for finite element analysis and structural design optimization is developed. The program uses database and the database management system MIDAS. The program is based on hypermatrix approach for assembly and solution of large matrix equations. Several example problems are solved using the program. An evaluation of data model, database and database management system in computer-aided structural design optimization environment is made. The performance of MIDAS in equation solving environment is determined using skyline and hypermatrix approach. Finally, it is concluded that with the proposed data models, database design methodology, and the advanced database management system, computer-aided design optimization of complex structural systems can be attempted.

Abstract approved

Jasbir S. Arora

Thesis supervisor

Civil & Environmental Engrg.

Title and department

September 17, 1985

Date



**COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION  
USING A DATABASE MANAGEMENT SYSTEM**

by

Thammaiah Sreekantamurthy

A thesis submitted in partial fulfillment of  
the requirements for the Doctor of Philosophy  
degree in Civil and Environmental Engineering  
in the Graduate College of  
The University of Iowa

December 1985

Thesis supervisor: Professor Jasbir S. Arora



Graduate College  
The University of Iowa  
Iowa City, Iowa

CERTIFICATE OF APPROVAL

---

PH.D. THESIS

---

This is to certify that the Ph.D. thesis of

Thammaiah Sreekantamurthy

has been approved by the Examining Committee  
for the thesis requirement for the Doctor of  
Philosophy degree in Civil and Environmental  
Engineering at the December 1985 graduation.

Thesis committee:

Jasbir S. Arora  
Thesis supervisor

Harinder G. J.  
Member

A. S. Agarwal Bhattach  
Member

Hareem K.  
Member

James R. Kelly  
Member



TO  
My parents and my brothers and sister



## ACKNOWLEDGEMENTS

First of all, I wish to express gratitude to my advisor and thesis supervisor Professor Jasbir S. Arora for his constant support and encouragement, and for his invaluable suggestion and advice. Professor Arora introduced me to the concept of optimal design of engineering systems and influenced the area of my research topic. It has been a pleasure and privilege to work with Professor Arora for the last three years. I also thank my committee members, Professor J.S. Arora, Professor H.C. Wu, Professor M.A. Bhatti, Professor H. Kane and Professor R. Shultz for the careful reading of the manuscript and helpful suggestions to improve this work. The financial support provided by the Graduate College of the University of Iowa, and the Air Force Office of Scientific Research is gratefully acknowledged. Finally, I am grateful to my wife, Jayashree S., for her understanding and patience and to Mrs. Peggy Duwa and to Mrs. Tina Swartzendruber for their excellent typing of the manuscript.



# TABLE OF CONTENTS

	Page
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
LIST OF SYMBOLS.....	xii
CHAPTER	
1. INTRODUCTION.....	1
1.1 Introductory Remarks.....	1
1.2 Computers in Structural Design - State-of-the-Art.....	2
1.3 Motivation for Research.....	5
1.4 A Survey of Literature.....	6
1.5 Objectives of Research.....	12
1.6 Scope of Work.....	13
2. COMPUTER-AIDED STRUCTURAL DESIGN.....	15
2.1 Introductory Remarks.....	15
2.2 Structural Design Process and a Sample Design Problem.....	15
2.3 Mathematical Modelling of Structural Design.....	19
2.3.1 Finite Element Analysis.....	19
2.3.2 Optimal Structural Design.....	23
2.4 Components Required to Develop Computer-Aided Structural Design System.....	26
2.5 Need for a Database in Computer-Aided Structural Design Optimization.....	29
2.6 Communication Subsystem.....	33
2.7 Users of Computer-Aided Structural Design System.....	35
3. A STUDY OF DATABASE MANAGEMENT CONCEPTS.....	38
3.1 Introductory Remarks.....	38
3.2 Problems Associated in Providing a Good DBMS for CAD-OPT.....	39
3.3 Definition of Various Terminologies.....	40
3.4 Views of Data Used in Strucutral Design and Data Models.....	49
3.4.1 Hierarchical Model.....	51
3.4.2 Network Data Model.....	51



	Page
3.4.3 Relational Model.....	53
3.4.4 Numerical Model.....	56
3.4.5 Choice of Data Model for Structural Design.....	58
3.5 Normalization of Data.....	59
3.6 Global and Local Databases.....	64
4. DATABASE DESIGN METHODOLOGY FOR STRUCTURAL ANALYSIS AND DESIGN.....	67
4.1 Introductory Remarks.....	67
4.2 Aspects Considered in the Proposed Methodology and Background.....	68
4.3 Methodology to Develop a Conceptual Data Model.....	72
4.3.1 Basic Considerations.....	72
4.3.2 Identification of Conceptual Data Objects.....	73
4.3.3 Reduction to Elementary Relations.....	79
4.3.4 Determination of Transitive Closure.....	84
4.3.5 Selecting Elementary Relations to Form a Conceptual Data Model.....	86
4.4 Methodology to Design an Internal Model.....	93
4.5 Some Aspects to Accommodate an External Model.....	103
4.6 Methodology to Incorporate Large Matrix Data into a Database.....	106
4.6.1 Identification of Matrices.....	106
4.6.2 Methodology for Design of a Numerical Model.....	108
4.7 An Algorithm or a Data Model?.....	120
5. DATABASE MANAGEMENT SYSTEM FOR STRUCTURAL DESIGN - A PROPOSAL.....	122
5.1 Introductory Remarks.....	122
5.2 Components Required in a Database Management System.....	123
5.2.1 Command Processor.....	124
5.2.2 Input-Output Processor.....	126
5.2.3 Addressing and Searching.....	127
5.2.4 File Definition and File Operations.....	128
5.2.5 Memory Management.....	129
5.2.6 Relational Operators.....	129
5.2.7 Security Schemes.....	130
5.3 Requirements of Data Definition Language.....	131
5.4 Requirements of Data Manipulation Language.....	134
5.5 Requirements of Query Language.....	136
5.6 A Review of Database Management Systems.....	138
5.7 Summary of Requirements of a DBMS.....	148



6.	IMPLEMENTATION OF A DATABASE MANAGEMENT SYSTEM -- MIDAS.....	152
6.1	Introductory Remarks.....	152
6.2	Implementation of MIDAS/R.....	153
6.2.1	Capabilities of MIDAS/R.....	154
6.2.2	Database of MIDAS/R.....	154
6.2.3	Data Definition Commands of MIDAS/R.....	155
6.2.4	Data Manipulation Commands of MIDAS/R.....	159
6.2.5	Interactive Commands.....	164
6.2.6	System Design of MIDAS/R.....	167
6.2.7	Limitations of MIDAS/R.....	169
6.3	Description of MIDAS/N.....	170
7.	A DESIGN OF DATABASE FOR STRUCTURAL ANALYSIS AND OPTIMIZATION DATA.....	173
7.1	Introductory Remarks.....	173
7.2	Identification of Data Used in Finite Element Analysis and Structural Design Optimization.....	173
7.2.1	A List of Entities of Analysis and Design Data.....	174
7.2.2	A List of Domains of Analysis and Design Data.....	176
7.3	Design of a Conceptual Data Model.....	179
7.3.1	Elementary Relations and Diagraph Representation.....	179
7.3.2	Deriving Additional Relations.....	180
7.3.3	Selecting Elementary Relations to Form a Conceptual Data Model.....	193
7.4	Design of Internal Data Model.....	196
7.4.1	Data Needed in Computation Process.....	197
7.4.2	Relations and Matrices for Internal Data Model.....	198
7.5	An External Data Model Design.....	205
7.6	Evaluation of Database Design Methodology.....	205
8.	IMPLEMENTATION OF A COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION SYSTEM USING DBMS.....	210
8.1	Introductory Remarks.....	210
8.2	Capabilities of the Program.....	211
8.3	System Design.....	212
8.3.1	DBMS Used in the Program.....	212
8.3.2	Finite Element Analysis Program.....	212
8.3.3	Design Sensitivity Analysis Program.....	220
8.4	Example Problems Solved Using the Program.....	224



	Page
9. EVALUATION OF DATABASE, MIDAS AND COMPUTER PROGRAM FOR STRUCTURAL DESIGN OPTIMIZATION.....	233
9.1 Introductory Remark.....	233
9.2 Evaluation of the Database used in the Program.....	233
9.3 Evaluation of DDL, DML and Query Language of MIDAS.....	241
9.4 Evaluation of MIDAS in Structural Design Applications.....	244
9.4.1 Skyline and Hypermatrix Approaches.....	245
9.4.2 Performance of MIDAS/R.....	247
9.4.3 Performance of MIDAS/N.....	249
9.4.4 Comparison of MIDAS/N and MIDAS/R Using Skyline Approach.....	251
9.4.5 Comparison of MIDAS/N and MIDAS/R Using Hypermatrix Approach.....	253
9.4.6 Performance of Memory Management of MIDAS/R.....	255
9.5 Evaluation of the Computer Program for Structural Design Optimization Using MIDAS.....	256
10. SUMMARY, DISCUSSION AND CONCLUSIONS.....	264
10.1 Summary	264
10.2 Discussion	268
10.3 Conclusions.....	272
10.4 Scope for Future Work.....	274
APPENDIX I. AN ALGORITHM FOR TRANSITIVE CLOSURE.....	276
APPENDIX II. BNF DESCRIPTION OF THE PROPOSED DATA DEFINITION LANGUAGE.....	278
APPENDIX III. BNF DESCRIPTION OF THE PROPOSED DATA MANIPULATION LANGUAGE.....	286
REFERENCES.....	292



## LIST OF TABLES

Table	Page
4.3.1 Transitive Closure for Elementary Relations.....	89
4.3.2 Deriving Minimal Cover.....	92
5.7.1 Features of Various Database Management Systems for Engineering Applications.....	151
9.4.1 Performance of MIDAS/R.....	248
9.4.2 Performance of MIDAS/N.....	250
9.4.3 Comparison of MIDAS/N and MIDAS/R Using Skyline Approach.....	252
9.4.4 Comparison of MIDAS/N and MIDAS/R Using Hypermatrix Approach.....	254
9.4.5 Performance of Memory Management of MIDAS/R.....	257
9.5.1 Summary of Evaluation Parameters for Ten Bar Truss.....	259
9.5.2 Summary of Evaluation Parameters for Twenty Five Bar Truss.....	259
9.5.3 Summary of Evaluation Parameters for Forty Seven Bar Truss.....	260
9.5.4 Summary of Evaluation Parameters for Seventy Two Bar Truss.....	260
9.5.5 Summary of Evaluation Parameters for One Hundred Eight Bar Truss.....	261
9.5.6 Summary of Evaluation Parameters for Two Hundred Bar Truss.....	261



## LIST OF FIGURES

Figure	Page
2.2.1 Conceptual Design of a Frame.....	17
2.2.2 Designer's View of a Frame.....	18
2.3.1 A General Flow Diagram for Optimal Design of Structure.....	27
2.4.1 Components of Computer-Aided Structural Design System.....	28
2.7.1 Users of Computer-Aided Structural Design System.....	37
3.3.1 Functional Dependencies.....	46
3.3.2 Full Functional Dependency.....	46
3.3.3 Transitive Dependence.....	46
3.3.4 Digraph.....	48
3.4.1 Hierarchical Data Model.....	52
3.4.2 An Occurrence of a Hierarchical Data Model.....	52
3.4.3 A Network Model.....	54
3.4.4 An Occurrence of a Network Model.....	54
3.4.5 Relational Model.....	55
3.4.6 Examples of Numerical Data Model.....	57
3.5.1 First Normal Form for Relation CONN.....	61
3.5.2 Second Normal Form for Relation CONN.....	63
3.5.3 Third Normal Form for Relation NAM-DOF.....	63
3.6.1 Network of Databases.....	66
4.3.1 Digraph Representation of Elementary Relations.....	87



Figure	Page
4.3.2 Connectivity Matrix C for Elementary Data.....	88
4.3.3 Digraph Representation of Minimal Cover.....	91
4.4.1 A Tentative Internal Model.....	96
4.4.2 Relation TRM-D in 1NF.....	96
4.4.3 Relations in 2NF.....	101
4.6.1 Banded Matrix.....	109
4.6.2 Hyper Matrix.....	109
4.6.3 Skyline Matrix.....	109
4.6.4 Row and Submatrix Storage Schemes.....	113
4.6.5 Relation for Matrix Storage.....	116
4.6.6 Transforming Internal Storage to External Views.....	118
4.6.7 Row-Column Storage Scheme (Internal).....	118
4.6.8 External View of Sparse Matrix.....	119
5.2.1 Components of a Database Management System.....	125
6.2.1 Data Type and Size of a Relation.....	157
6.2.2 Layout of Data in a Typical Relation.....	158
7.3.1 Digraph Representation of Association Between Entities and Attributes.....	181
7.3.2 Initial List of Elementary Relations.....	187
7.3.3 Initial Connectivity Matrix.....	188
7.3.4 Final Connectivity Matrix.....	189
7.3.5 Derived Elementary Relations.....	190
7.3.6 Transitive Closure for the Elementary Relations (Partially Shown).....	194
7.4.2 Relations for Internal Model.....	199



Figure		Page
7.4.3	Relations for Internal Model.....	204
7.5.1	Relations for External Model.....	206
8.3.1	Modules in Finite Element Analysis Program.....	214
8.3.2	Assembled Stiffness Hypermatrix.....	217
8.3.3	Load Hypermatrix.....	219
8.3.4	Modules in Design Sensitivity Analysis Program.....	222
8.4.1	Ten-Bar Truss.....	226
8.4.2	Twenty-five-Member Transmission Tower.....	227
8.4.3	Forty-seven-Member Plane Truss.....	228
8.4.4	Seventy-two-Member Space Truss.....	229
8.4.5	Two Hundred-Member Plane Truss.....	230
8.4.6	Geometry of Helicopter Tailboom.....	231
8.4.7	Arrangement of Members for Open Truss Helicopter Tailboom.....	232



## LIST OF SYMBOLS

$\bar{A}$	Element cross-sectional property matrix
$B$	Strain-displacement matrix
$b, b_i$	Design variable value
$b_{il}$	Lower limit of design variable value
$b_{iu}$	Upper limit of design variable value
$C$	Damping matrix of a structure
$D$	Constitutive matrix
$F$	Body force
$F_e$	Element forces
$F^{t+\Delta t}$	Vector of nodal forces equivalent to element stresses
$h$	Equilibrium equation
$K_{bb}$	Assembled boundary stiffness matrix
$K$	Assembled stiffness matrix of a structure
$K_b$	Condensed stiffness matrix of a substructure
$K_{bi}$	Assembled internal-boundary stiffness matrix
$K_e$	Element stiffness matrix
$K_{eff}$	Effective stiffness matrix of a structure
$K_{ii}$	Assembled internal stiffness matrix
$K_{NL}$	Nonlinear geometric stiffness matrix
$K_L^t$	Linear strain incremental stiffness matrix
${}^tK$	Tangential stiffness matrix



$K_g$	Geometric Stiffness matrix
$M$	Assembled mass matrix of a structure
$M_e$	Element mass matrix
$N$	Shape function matrix
$n$	Element number
$P$	Global loads on a structure
$P_B$	Element body force
$P_b$	Boundary loads of a substructure
$P_b^*$	Condensed load matrix of a substructure
$P_e$	Element load vector
$P_{eff}$	Effective load matrix of a structure
$P_i$	Internal loads of a substructure
$P_s$	Element surface loads
$P^{t+\Delta t}$	Vector of externally applied loads at time $t+\Delta t$
$P_0$	Element load due to initial strain
$r$	Substructure number
$T$	Transformation matrix
$U$	Global displacement of a structure
$u_e$	Element Displacements
$u_i$	Internal displacements of a substructure
$u_b$	Boundary displacements of a substructure
$V$	Volume
$y$	Mode shape
$z$	Displacement vector
$z_j^a$	Allowable displacement at $j$ th location



$z_j$	Displacement of $j^{\text{th}}$ location
$\Delta \mathbf{v}$	Vector of incremental nodal displacements
$\epsilon_e$	Element strain
$\epsilon_0$	Initial strain
$\rho$	Material density
$\sigma_a$	Allowable stress in the member
$\sigma_b$	Buckling stress of a member
$\sigma_e$	Element stress
$\sigma_{ij}$	Stress in the member
$\lambda$	Eigenvalue
$\lambda_i$	Adjoint matrix
$\psi$	Constraint function
$\psi_0$	Cost function
$\zeta$	Frequency
$\zeta_0$	Lower bound on eigenvalue



## CHAPTER 1

### INTRODUCTION

#### 1.1 Introductory Remarks

Advances in computer technology have brought about profound changes in the way engineering analysis and design are performed. In structural analysis computer has become a vital adjunct to theory. Several general purpose computer programs having a wide range of capabilities are currently in use for finite element analysis of structures. In the structural design field, computer programs are being developed for optimal design. But, they are in the early stage of development and are facing a number of problems in design of practical structures. Problems arise due to iterative nature of optimal design algorithms and need for reanalysis of the structure in each iteration. Reanalysis of a structure using existing finite element programs are difficult because they are not flexible to use modified data generated at various design stages. Moreover, designer needs control over the program and data in selecting appropriate algorithms and data to obtain optimum solution. Thus, there exists a wide gap between structural analysis and design capabilities. To bridge this wide gap, it is necessary to establish approaches through integration of computers in the design environment. The term computer-aided design has evolved over years which provide a good basis for such an integration.



Computer-aided design (CAD) means integration of engineering methods and computer-science in a computer-based system, using a database, a program library and a man-machine communication link. The term computer-aided design optimization (CAD-OPT) is derived from the above definition to cover analysis and design optimization methods. In this study a new concept is presented for integrating structural analysis and design optimization methodology into a computer-based system which encompasses this meaning of CAD. Emphasis is placed upon database management concepts for finite element analysis and structural design optimization.

### 1.2 Computers in Structural Design - State-of-the-Art

Knowledge about the historical background provides a better understanding of the state-of-the-art. Going back to 1960, Integrated Civil Engineering Systems (ICES) development was an important milestone in the use of computers in solving civil engineering problems. It was based on the idea of integrating computers in the problem solving environment to provide faster, more accurate and complete analysis and design capability to engineers. During the same period, theory of the finite element method began to evolve and led to the development of several finite element analysis programs. The computer programs such as NASTRAN, STRUDL, and ASKA are well known and widely used for finite element analysis. Many of these programs are quite sophisticated, large in size and are capable of analyzing a wide variety of structural problems. With the advances in computer technology, computers are



available at a lesser cost and have additional facilities like graphic display, and large disk capacities. Finite element programs were designed to make use of such facilities to display finite element mesh, store large amounts of data on disk and provide interactive facility to users. Programs like GIFTS, ANSYS, and ADINA were developed in the seventies to provide these new features to users.

During the last decade, research activity in the area of structural design optimization increased. Investigations on nonlinear programming techniques in structural optimization became one of the major topics of research. Several computer programs were developed for solving structural optimization problems. These include DOCS (Arora, et al., 1984a), ACCESS (Fleury, et al., 1981), PROSSS (Sobieszczanski-Sobieski, et al., 1980) ODYSSEY (Bennett, 1979) and others having moderate range of capabilities in solving optimization problems. They use finite element method for analysis of structures. DOCS program has capability to use substructuring, design damaged structures, and to use gradient-based techniques for optimization. PROSSS uses SPAR program for finite element analysis. Many of these programs were developed to study problems of research interest. Therefore, applicability of the programs to general structural problems is limited. None of these programs is linked to any pre- or post-processor making input to program and analysis of results extremely difficult. Studies are being made to develop good structural design optimization programs that are comparable to the generality and capabilities of existing finite element programs.



Since, finite element analysis of structures uses large amount of data, some routines were incorporated into finite element packages to store data on secondary storage devices. Data management using these routines was tedious and applicable only to one program. Data generated by finite element packages were almost impossible to use in other programs for further analysis and design of structures. Development of design optimization algorithms, faced this problem for using analysis data generated by finite element packages. Iterative design process posed a big challenge not only in efficient use of computer resources, but also in organization of a large amount of data of finite element analysis and design optimization methods. At this stage, engineering software designers began to think of introducing database management concepts into the software similar to those of business database management systems. Several database management programs were developed in the late seventies and in the beginning of the eighties for engineering applications. Integrated programs for Aerospace Vehicle Design (IPAD) development is an important milestone in engineering database management. A database management system called RIM (RIM, 1982) was developed under IPAD project. Several application programs such as SPAR (Giles and Haftka, 1978), BANDIT (band width minimization), PROSS, ATLAS, and NPLOT (graphics) were tied together with a common database for integrated design of structural systems (Fishwick and Blackburn, 1982). However, use of a database in these application programs was limited to input and output only. There does not exist a finite element program which directly uses a generalized database



management system such as the one developed for IPAD project. Studies are being made to incorporate a database, a program library and man-machine communication link as needed for computer-aided structural design. Emphasis on blending computer-science and engineering methodology toward arriving at efficient and economic design of structural systems seems to be the goal of CAD today.

### 1.3 Motivation for Research

In optimal design of structural systems we generally use nonlinear programming and finite element techniques. Nonlinear programming techniques require formulation of design objectives and constraints of the system. They use large amount of data depending on the size and complexity of problem. Organization of data related to design variables, geometry, material, loads, and intermediate computation data, generated and used in design of large structural systems is a tedious task. Finite element techniques are usually adopted to analyze the system within a design iteration. As such the finite element techniques require huge amount of computation and data storage depending on the size of problem at hand. Further, the amount of data handled depends directly on the number of iterations performed in iterative design optimization algorithms. Therefore, there is a need for data organization in optimal design of structural systems.

In this regard, incorporating a database into structural design programs looks attractive. Such a database can provide data for both structural design optimization and finite element analysis programs. It



will enable designers to choose appropriate data from the database and use them in any optimization algorithms to improve design. Also, data used and generated in one program can be made available for use in another program. Since, most of the data for finite element analysis and design optimization are common, a centralized database will provide efficient organization of computer resources. A centralized database allows interaction between a finite element program and an optimization program to improve design iteratively. Such a database will provide an option for the designer to interrupt the program execution and provide flexibility for the designer to change the design parameters. A good database will enable addition of new optimization and other programs which use the common data without extensive modification of database or existing programs. Also, several designers can be allowed to use a common database to investigate alternate designs. Interactive graphics data can be stored in a database to provide easy communication between computer and designer. Therefore, a properly designed database, together with a set of design programs and communication system, offer a considerable aid to engineers involved in design optimization.

In view of the above observation, we will investigate design and use of a database and a database management system in structural design optimization.

#### 1.4 A Survey of Literature

A survey of literature of data management in computer-aided structural design is given in this section. The survey also includes



literature on database management for engineering applications. The survey is broadly classified into database management concepts and systems. Various database management systems currently in use are reviewed in Chapter 5 and their features tabulated there.

The meaning of computer-aided design has changed several times in the past two decades of its usage. It was a popular idea that CAD meant a menu of analysis programs called by the designer. Later, CAD became synonymous with computer-aided drafting. However, a true description of CAD is synergistic interplay of man and computer (Allan 1972). A more appropriate definition of CAD is given by Encarnação and Schlechtendahl (1983): "It is a discipline that provides know-how in computer-software and hardware in system analysis and in engineering methodology for specifying, designing, implementing, introducing, and using computer based systems for design purpose."

The paper by Felippa (1979) serves as an introduction to the subject of database management for scientific and engineering applications. It highlights the differences between the business data management and scientific data management. A comprehensive list of terminology relevant to scientific computing is given in another paper by the author (Felippa, 1980). Since, the terminology used in business DBMS is fairly new to engineers, this list serves as a starting point for the newcomers. It is interesting to know how scientists actually use their data. The paper by Bell (1982) discusses some issues about data usage and also gives comparison between data modelling for scientific and business applications. In order to bring out differences



between the use of database for business and engineering applications, Foisseau and Valette (1982) describe a list of criteria.

The application of data management in finite element analysis and design optimization computation is fairly new. Even though, data management in these computations is critically needed, not much attention has been paid to develop proper data management techniques. Only a few research studies were made on data management for finite element analysis. Lopez (1978) and associates studied the application of data management to structures. They pointed out that serious drawbacks of ASKA, STRUDL and NASTRAN were due to lack of high level database management facility. Also, these programs did not provide any type of data structure capability. They have simple internal organization and require many files with trivial data structures. These type of system tend to be I/O bound because logical operations on data are related directly to a physical location on a serial device. For example, in generating the stiffness matrix for the elements of a structure, most programs generate one matrix and write onto a sequential device; and the process is repeated for all elements of a structure. In order to access these stiffness matrices at a later time, the program must pass serially over the entire file again. Lopez (1974) in another paper presented a data management system for finite element analysis. Pahl (1981) described the properties and functions of data storage for finite element programs.

Several research studies were made on data management techniques for computer-aided design and general engineering applications. The



techniques developed for them are also applicable for finite element analysis and design optimization. Studies on data models, database design methodology, database network, data definition language, data manipulation language, database integrity and consistency and numerical database management were conducted by several researchers. A comprehensive survey of data management in engineering applications is given by Sreekanta Murthy and Arora (1985a).

The well-known data models -- hierarchical, network and relational have been studied by many researchers to find out their suitability for organizing engineering data. Koriba (1983) discusses applicability of ANSI/SPARC, CODASYL and relational approach to CAD software design. The three levels of data view proposed by ANSI/SPARC is gaining wide acceptance and is likely to be incorporated into future CAD systems. Relational approach is based on set concepts and provides a sound mathematical background. This approach provides high level of data independence, user friendly data definition and data manipulation capabilities. Relational model is becoming popular among database designers and users. Several researchers are currently working on this model. Fishwick and Blackburn (1982) discuss advantage and disadvantage of a relational model from an engineering point of view. Authors provide examples of relations for managing data of a finite element model. They also described the development of the PRIDE system which integrates engineering application programs -- AD-2000, SPAR, PROSSS, NCAR, and BANDIT. SPAR and PROSSS are finite element analysis and structural optimization packages respectively. AD-2000 is a finite



element model generator and BANDIT is bandwidth optimization program. However, use of their database management system was limited to interfacing application program input and output to a common database. Modification of application programs was not made to use the database for programs internal data organization needs. Blackburn, Storaasli and Fulton (1982) in another paper demonstrate the use of a relational database in engineering applications. Four sample problems -- a panel with circular hole, a square plate, a conventional wing structure and a large area space structure were used to evaluate the merits of managing engineering data using a relational system. Studies on hierarchical model are mainly with reference to organizing large matrices. Lopez (1974) uses a hierarchical model for finite element data organization. A hierarchical data structure for organizing node, element, load and stiffness matrix data is given in the paper. However, the DBMS uses a problem-oriented language translating facilities in the POLO supervisor under which the application programs operate. Hence, it is highly doubtful that two will ever be used independently of each other. Pahl (1981) described hierarchical storage structure for hypermatrix data organization. Hypermatrix stiffness and load data are found to be most suitable to hierarchical data representation. A paper by Elliot, Kunni, and Browne (1978) describes a hierarchical model of data and a DBMS system design based on it. Some practical examples on structural design and wind tunnel data management are also given in the paper. But, this system requires a precompiler to decode the data description and data manipulation commands in a source program.



Investigations have been conducted to find out a suitable way to design a database for engineering applications. There exists basically two different approaches to database design -- first approach generates a global schema and then derive local views from it; the second one obtains local views of different users and then integrate them to form a global view. Buchmann and Dale (1979) analyze different methodologies to design a database and present a frame work for evaluating them. A comprehensive description of database design methodology for business applications is given in Vetter and Maddison (1983). Several researchers, Lillehagen and Dokkar (1982), Grabowski, Eigener and Ranch (1978), and Eberlein and Wedekind (1982) have worked on database design for CAD applications. So far, there does not exist any methodology to design database of finite element and design optimization programs.

Development of suitable data definition (DDL) and data manipulation languages for engineering applications have been of interest to many researchers. One of the major considerations in the design of data definition language was to keep the syntax concise and easy to use for application programmers. Several other important considerations in DDL design are described in detail by Elliot, Kunni and Browne (1978). They use special indicators in the source program code to identify the DDL and DML commands and translate them using a precompiler to FORTRAN statements. These DDL and DML statements can be used to operate on a hierarchical data structure. Special features of DDL and DML in a relational DBMS for interactive design are described by Shenoy and Patnaik (1983).



The application of data management in numerical computation is fairly new. Finite element analysis and design optimization procedures require substantial amount of matrix data processing. Data management system require special facilities to deal with data of large matrices. A recognition of this need is made by Daini (1982) and a model is developed for numerical database arising in many scientific applications to keep track of large sparse and dense matrices. The paper describes a generalized facility for providing data independence by relieving users from the need for knowledge of physical data organization on the secondary storage devices. Because of the limitation of core storage and to reduce the input-output operations involved in secondary storage techniques, many investigations have been conducted on the efficient use of primary memory. A detailed survey by Pooch and Nieder (1973) gives various indexing techniques that can be used in dealing with sparse matrices. Darby-Dowman and Mitra (1983) describe a matrix storage scheme in linear programming. Rajan and Bhatti (1983) described a memory management scheme for finite element software. Sreekanta Murthy, Reddy and Arora (1984) describe the database management concepts that are applicable to design optimization field.

### 1.5 Objectives of Research

1. To study of various database management concepts applicable to computer-aided structural design optimization field. Suitability of available database management concepts and drawbacks associated with their use in engineering design will be investigated.



2. To develop a suitable database design methodology for structural design database and to develop a conceptual data model to represent the design data. Schemes for constructing internal and external data models will be identified. Data models for organizing matrix data will be developed.
3. To study the existing database management systems and to identify the important features with respect to their suitability to organize structural design data.
4. To formulate requirements of data definition language, data manipulation language and query language for engineering design database management system.
5. To implement a database management system for engineering applications based on selected data models.
6. To design a database for structural design optimization. Use of database design methodology for constructing conceptual, internal and external model will be demonstrated.
7. To implement a computer-aided structural design optimization program using a database management system and evaluate its performance.

### 1.6 Scope of Work

Computer-aided structural design process is identified in Chapter 2. Mathematical modelling for finite element analysis and structural design optimization are given. Need for database management in structural design is stressed. Chapter 3 deals with database management concepts. Well-known data models are described with



reference to structural design data. Various database management concepts like normalization of data, and global and local databases are described. Database design methodology for structural design is given in Chapter 4. Methodology for conceptual model development for structural design database is described. Normalization procedures for developing internal data model are described with examples. Description of a proposed numerical model is given there. In Chapter 5, requirements of a database management system are studied. Requirements of data definition and data manipulation languages for structural design database are formulated. Considerations in developing memory management schemes and query language are presented. Implementation details of a database management system for organizing structural design data are described in Chapter 6. Relational data management procedures and numerical data management schemes are described. Usefulness and drawbacks of this systems are given. In Chapter 7, database design for structural optimization is described. An evaluation of database design methodology is given. Computer program developed for finite element analysis and structural design optimization is described in Chapter 8. The capabilities of the program and example problems solved using it are given there. In Chapter 9, evaluations of the database, the database management system, and the computer program for structural optimization are made. Finally, discussion and conclusions of the present study are given in the last chapter.



## CHAPTER 2

### COMPUTER-AIDED STRUCTURAL DESIGN

#### 2.1 Introductory Remarks

In this chapter, the principles, methods and tools required to develop a computer-aided design of structural system are described. Structural design process is described with the aid of a sample design problem to provide qualitative description of the design process. In particular, mathematical modelling of the structural design process is given in Section 2.3 to bring out various steps. As mentioned in Chapter 1, a database, a program library and a communication link form the important components of a CAD system. These components are described in detail in Section 2.4. Need for data management for structural design is emphasized in Section 2.5. Need for a good communication subsystem is given in Section 2.6. Finally, various classes of users of a computer-aided structural design system and their requirements are identified in the last section.

#### 2.2 Structural Design Process and a Sample Design Problem

A study of overall structural design process is necessary to develop a computer-aided structural design system. The design process can be described, in general, by a sequence or chains of actions where each action passes its results on to its successors. The complex nature



of design process will have to be reflected in CAD systems if such systems are to support the design process as a whole.

The design process begins with the identification of a need by a user of the structural system. The needs and objectives of the system are defined quantitatively. Functional analysis is carried out to find out operational requirements of the structural system. The next step is the configuration or conceptual design of the system. For example, if function to be performed is to support the loads on a frame, the conceptual design (see Fig. 2.2.1) includes beams, columns, plates, and bars. At the conceptual design stage various parameters describing the system are identified and acceptable range of values are prescribed. This preliminary design is then analyzed with respect to the constraints and if it does not adequately satisfy the constraints, the design is revised. This is an optimal design process, which has its objectives the choice of undetermined parameters that were identified in the previous step. The criterion for optimal design may be maximization of structural system capability or minimization of cost. The analysis and redesign cycles are repeated until a design satisfying all the constraints is obtained.

It is common that a number of designers working on a practical design project carry out specific subtasks. Designers are required to meet individual goals, and may have an isolated view of the project. For example, designer A (see Fig. 2.2.2) may work only on part of the structure in the design project. During the process, a set of information required for individual needs is derived to carry out the



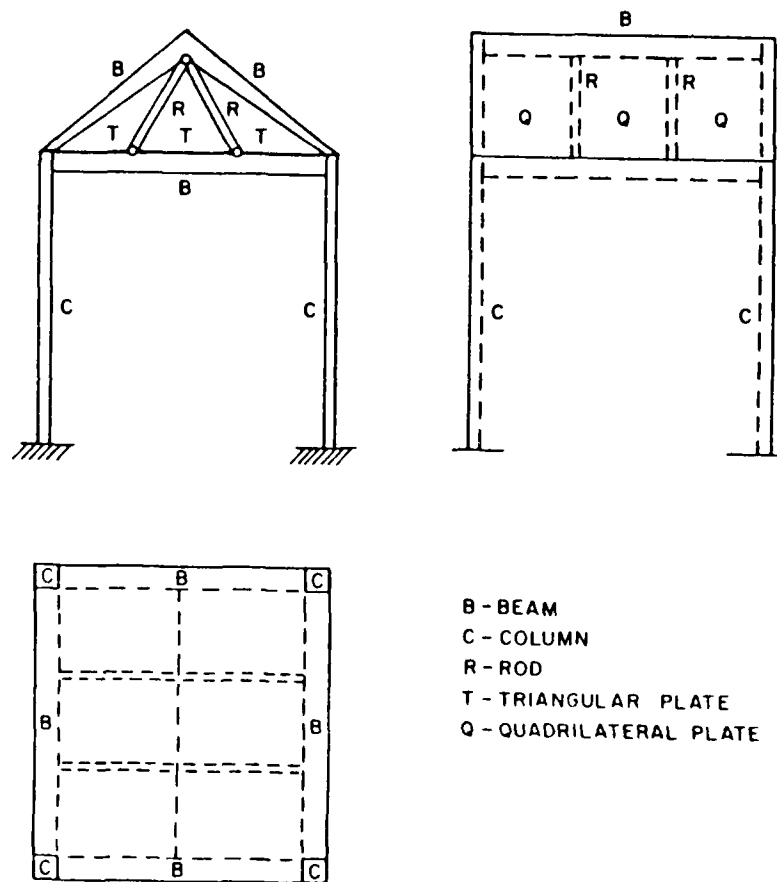


Figure 2.2.1 Conceptual Design of a Frame



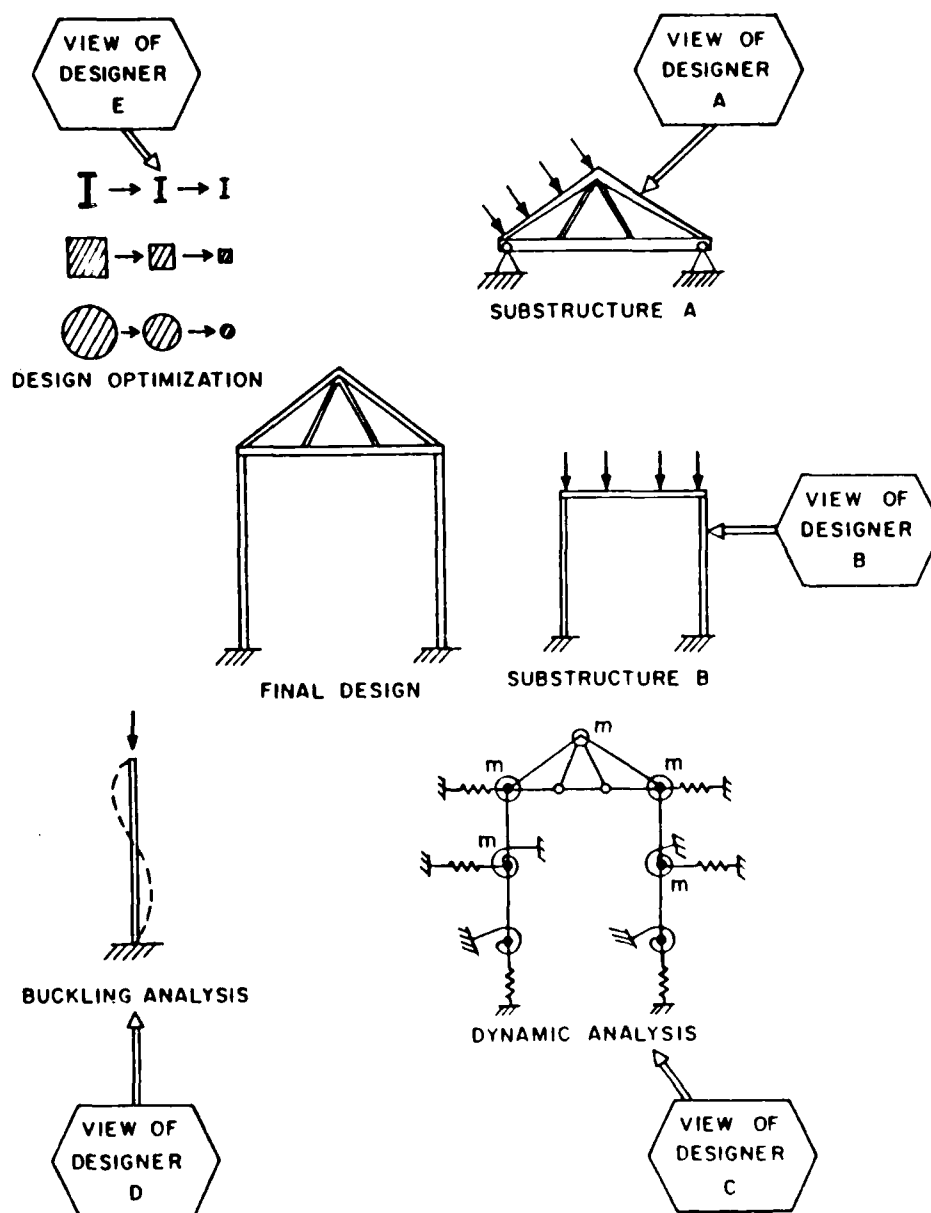


Figure 2.2.2 Designer's View of a Frame



specific task. The sharing of information takes place, between the conceptual design level with subsystem design levels, and among subsystem design levels themselves. The method of information sharing (reports, catalogues, etc. in case of conventional design process; a database in case of CAD design process) and tools for information-sharing (typing, printing, drafting in conventional design; interactive computer terminals, graphics in CAD design process) are dependent on the state-of-art of the design process.

### **2.3 Mathematical Modelling of Structural Design**

In the previous section, a general structural design process was described. Here, mathematical modelling for structural design is given. This mathematical model is intended to provide a basic framework for developing a computer-aided structural design system. Various steps of the structural design are formulated in terms of mathematical models. Namely, identification of objectives and constraints, analysis of the structure by the finite element method, design constraint checks, design sensitivity calculations, and design optimization process are described.

#### **2.3.1 Finite Element Analysis**

Finite element analysis begins with idealization of the structure using a number of finite elements. The input data for a finite element analysis program consists of the geometric idealization, the material properties, and the loading and boundary conditions. Important steps of



finite element analysis (Cook, 1981) are listed below. Depending on the type of analysis, some or a combination of the steps are used.

**Element Level Computation.** At the element level, stiffness matrices, mass matrices, load matrices are computed as

$$K_e = \int_V B^T D B \, dV$$

$$M_e = \int_V N^T \rho N \, dV$$

$$P_e = P_o + P_s + P_B$$

$$P_o = \int_V N^T D \epsilon_o \, dV$$

$$P_s = \int_A N^T p \, dA$$

$$P_B = \int_V N^T F \, dV$$

**Substructure Level Computation.** If substructures are used in the idealization, then element stiffness matrices are assembled to form substructure level matrices. The equilibrium equation for the  $r$ th substructure is given by

$$K^r U^r = P^r$$

i.e.



$$\begin{bmatrix} K_{ii} & K_{bi} \\ K_{ib} & K_{bb} \end{bmatrix}^r \begin{bmatrix} U_i \\ U_b \end{bmatrix}^r = \begin{bmatrix} P_i \\ P_b \end{bmatrix}^r$$

The following computations are done

$$K_b^r = K_{bb}^r - K_{bi}^r \cdot K_{ii}^{-1r} \cdot K_{ib}^r$$

$$P_b^{*r} = K_{bi}^r \cdot K_{ii}^{-1r} \cdot P_i^r$$

$$K_{eff} = \sum_r K_b^r$$

$$P_{eff} = -\sum_r P_b^{*r} + P_b$$

**Structure Level Computation.** Equations of equilibrium for the complete structure are solved to get response of the structure. For static equilibrium, we have

$$KU = P$$

where

$$K = \sum_n K_e, \quad P = \sum_n P_e$$

For dynamic response of a linear structural model.



$$M\ddot{U} + C\dot{U} + KU = P$$

where

$$M = \sum M_e, \quad C = \sum C_e, \quad P = \sum P_e$$

For nonlinear analysis

$$(K_L^t + K_{NL}^t) \Delta U = P^{t+\Delta t} - F^{t+\Delta t}$$

For buckling analysis

$$(K + \lambda K_\sigma) U = 0$$

For frequency analysis

$$(K + \lambda M) U = 0$$

**Recovery of Element Level Response.** After the structure level response have been computed, element level displacements, stresses, strains, and forces are computed

$$u_e = TU$$

$$\epsilon_e = Bu_e$$



$$\sigma_e = BDu_e$$

$$F_e = \bar{A}\sigma_e$$

### 2.3.2 Optimal Structural Design

Optimal structural design is carried out using well-known methods given in Haug and Arora (1979) and Arora and Govil (1977). Important steps of optimal design consist of formulation of cost and constraint functions, checking for constraint violations, design sensitivity analysis, design change computations and convergence checks. These steps are listed below.

**Problem Formulation.** Optimal structural design problem is formulated using a set of state and design variables. The objective is to minimize the cost function

$$\psi_0(z, \zeta, b)$$

subject to state equations

$$h(z, b) = 0$$

$$K(b)y = \zeta M(b)y$$

and constraints

$$\psi(z, \zeta, b) < 0$$



**Constraint Checks.** Violated displacements, stress, frequency and other constraints are identified. For displacement constraints

$$\psi_i \equiv z_j - z_j^a < 0$$

For eigenvalue constraint

$$\psi_i \equiv \zeta_0 - \zeta < 0$$

For stress constraints

$$\psi_i \equiv \sigma_{ij} - \sigma_a < 0$$

For buckling constraint

$$\psi_i \equiv \sigma_{ij} - \sigma_b < 0$$

For design variable constraints

$$\psi_i \equiv b_i - b_{iu} < 0, \text{ or } b_{il} - b_i < 0$$

**Design Sensitivity Analysis.** Design sensitivity analysis is done to determine the effect on the problem functions of a change in design  $\delta b$  in  $b^0$ . Gradients of function  $\psi$  is



$$\frac{d\psi_i}{db} = \frac{\partial\psi_i}{\partial b} + \frac{\partial\psi_i}{\partial z} \cdot \frac{dz}{db} + \frac{\partial\psi_i}{\partial \zeta} \cdot \frac{d\zeta}{db}$$

The following computations are needed to calculate gradients:

1. Linear systems

$$K \frac{dz}{db} = - \frac{\partial h}{\partial b} \quad \text{for direct differentiation method}$$

$$K^T \lambda_i = \frac{\partial\psi_i}{\partial z} \quad \text{for adjoint variable method}$$

$$\frac{\partial h}{\partial b} = \frac{\partial}{\partial b} (K(b)\tilde{z} - F(b))$$

2. Nonlinear systems (Ryu, Haririan, Wu and Arora, 1984)

$$t_K^T \lambda_i^j = \frac{\partial\psi_i^T}{\partial z}$$

3. Sensitivity analysis of eigenvalues

$$\frac{\partial \zeta_j}{\partial b} = \mathbf{y}_j^T \left( \frac{\partial K}{\partial b} - \zeta \frac{\partial M}{\partial b} \right) \mathbf{y}$$

**Design Change Computation.** A change in design variable vector  $\mathbf{b}$  is computed to reduce the cost. Mathematical programming methods such as the gradient projection or other methods (Arora, et al., 1984b) are used to compute design change  $\delta \mathbf{b}$ :

$$\mathbf{b}^{v+1} = \mathbf{b}^v + \delta \mathbf{b}^v, \quad v = 0, 1, 2, \dots, \text{iterations}$$



A general flow diagram for optimal design of the structure is given in Fig. 2.3.1.

#### 2.4 Components Required to Develop Computer-Aided Structural Design System

For developing a computer-aided structural design system three important components -- a database, a program library, and a communication subsystem are required. A database contains data required for finite element analysis and structural design optimization. Several users operate on the database either interactively or through application programs. A database acts as a central repository of data for CAD applications. The second component, namely, a program library contains both the modules used for data management and modules containing algorithms needed for structural analysis and design applications (matrix operation library, equation solvers, finite element programs and optimization routines). Data management programs need basic components -- file management, input-output processor, memory management, addressing and searching, and security routines. Finally, a communication subsystem is needed to provide link between the computer and designer. They provide channels of data communication between the database, database management, and application programs. A communication subsystem consists of interactive command processors, data definition language, data manipulation language, and routines for graphic display. The basic components of a computer-aided structural design system are schematically shown in Fig. 2.4.1.



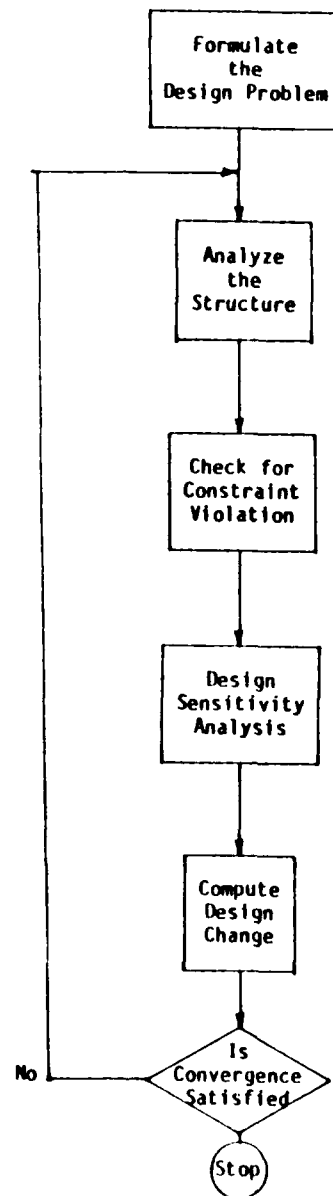


Figure 2.3.1 A General Flow Diagram for Optimal Design of Structure



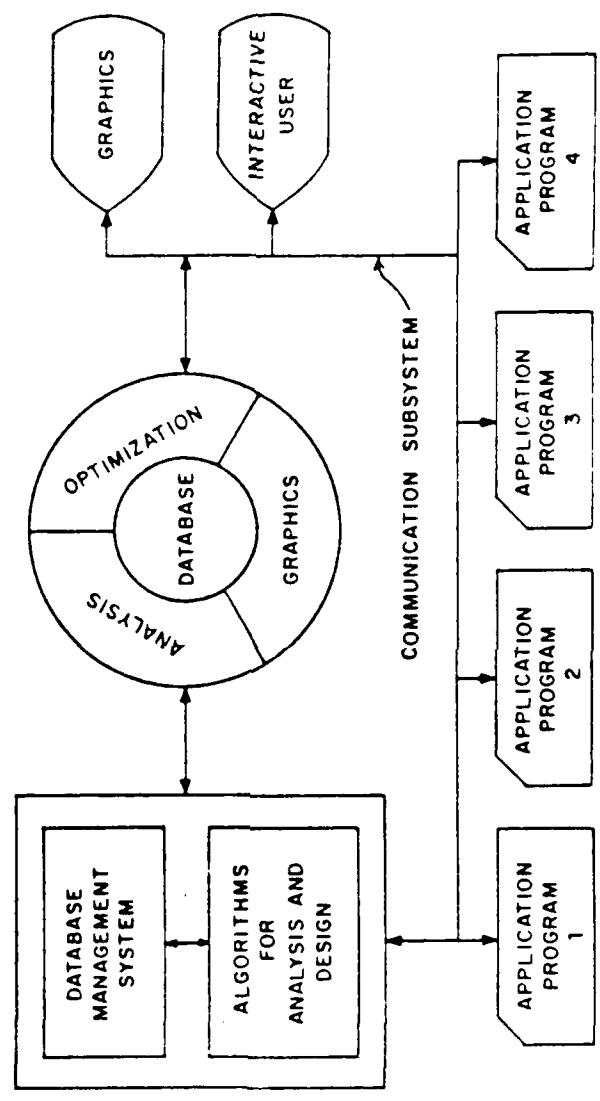


Figure 2.4.1.1 Components of Computer-Aided Structural Design System



Thus, we need to design and develop the three components -- a database, a program library and a communication subsystem to provide an efficient and economic means of designing a structural system using computer. In this study, database which is the most important component of the system is considered in detail.

### 2.5 Need for a Database in Computer-Aided Structural Design Optimization

In optimal design of structural and mechanical systems, we generally use nonlinear programming techniques (Haug and Arora, 1979). The design objectives and constraints for the system are described in a mathematical model. Design of a system is specified using a set of parameters called design variables. The design variables depend on the type of optimization problem. In design of aircraft components such as stiffened panels and cylinders, the design variables are spacing of the stiffeners, size and shape of stiffeners, and thickness of skin. In optimization of structural systems such as frames and trusses of fixed configuration the sizes of the elements are design variables. Thickness of plates, cross-sectional areas of bars, moment of inertia represent sizes of the elements. If shape optimization is the objective, the design variables may include parameters related to geometry of the system.

The constraints for the system are classified into performance and size constraints. The performance constraints are on stresses, displacements, and local and overall stability requirements in the static case; frequencies and displacements in the dynamic case; flutter



velocity and divergence in aeroelastic case, or a combination of these. The size constraints are the minimum and maximum value of design variables. In nonlinear programming, the search for the optimum design variable vector involves iterative schemes. The design variable data at the  $n$ th iteration is used to compute a direction vector and a step size along it. The direction vector involves computation of gradients of objective and constraint functions with respect to the design variables. Data belonging to equivalent design variables are grouped there by reducing the size of design variable vector.

In most problems of structural and mechanical system design, behavior of the system can be defined using state variables, e.g., stresses, displacements, and other response variables. In such a case, state space formulation is frequently employed (Haug and Arora, 1979). Design sensitivity coefficients in terms of matrix equation are determined in state space formulation. Adjoint equations are used to define a set of variables that provide design sensitivity information. Symmetric matrix equations can be used to advantage thereby reducing the data storage requirements.

Finite element and other numerical methods are used for analysis of structural and mechanical systems. Finite element method uses data such as element number, nodal connectivity, element stiffness matrix, element mass matrix, element load matrix, assembled stiffness, mass and load matrices, displacement vectors, eigenvalues, eigenvectors, buckling modes, decomposed stiffness matrix, and the stress matrix. In general data used in finite element and other numerical analysis procedures is



quite large. Symmetry of stiffness and mass matrices is taken into account so that data storage requirement is reduced. Hypermatix or other special schemes are generally used in dealing with large matrix equations.

For design of large structures, efficient design sensitivity analysis is particularly critical. For such structures, substructuring concept can be effectively integrated into structural analysis, design sensitivity analysis, and optimal design procedures (Haug and Arora, 1979). In this concept, one deals with small order matrices as the data can be organized substructure-wise. The degrees of freedom can be classified into boundary degrees of freedom and interior degrees of freedom. Data for the stiffness matrices corresponding to these degrees of freedom can be separately stored. Data of constraint functions corresponding to internal and boundary degrees of freedom are used in determining design sensitivity calculations. Adjoint matrix data is stored for each substructure.

Many real world problems will have features that are not explicitly contained in general optimal design formulation. Problems with peculiar features need to be treated by making minor alterations in the general algorithm. Interactive computation and graphics can be profitably employed in design optimization. At a particular iteration, the designer can study the data of design variables, constraints which are active, performance of the system, cost function, admissible direction of travel, sensitivity coefficients, etc. He can make judgement regarding suitability of a particular algorithm, change of system



parameters, and redefine convergence parameters to achieve optimal design. Interactive graphics requires additional data for display of system model, results, and graphs.

Thus, for design optimization, data generated during analysis must be saved in the database. This data is used for formulation of constraints. Constraints are checked for violation. Design sensitivity analysis of violated constraints is carried out using most of the data generated during analysis. Once design sensitivity analysis has been completed, a direction finding problem is defined and solved. Note that the size of direction finding problem at each iteration depends on the number of active constraints. Therefore, sizes of data sets change from iteration to iterations. Thus, the nature of data is quite dynamic. We should be able to dynamically create large data sets, manipulate them during the iteration, and delete some of them at the end of iteration. Useful trend information from each iteration must be saved for processing in later iterations. Note that a row of the history matrix (such as design variable values) is generated at each iteration. However, to use the trend information for a quantity (e.g., a design variable), we need to look at its value at the previous iterations. This implies that we should look at a column of the history matrix. Therefore, we should be able to create data in one form and view it another. Thus, we must have an intelligent and sophisticated DBMS. Data must be organized, saved in a database, and properly managed for design optimization.

We observed that large amount of computation is needed in the design process. Finite element analysis and optimization programs



generate large amount of data depending on the size of the problem. The amount of data used during design optimization stage depends directly on the number of iterations. Several application programs are used during the design process and each of them requiring specific data. Several algorithms may be needed which use similar data to arrive at optimal design. In such a case data used by one algorithm should be made available for use in another algorithm. Therefore, designer needs control to select appropriate algorithm and data to obtain optimum solution. An important feature of design database is that it contains both informative data such as geometry, material property as well as operational data such as stiffness matrix. Informative data remains static where as operational data gets continuously updated, modified and deleted. A centralized database is needed which stores all the data of analysis and design. A centralized database provides an option for the designer to interrupt the program execution and provides flexibility for the designer to change the design parameters. Therefore, a careful consideration of data organization in a database is necessary to improve design efficiency.

In summary, we have identified the need for a database for structural design and special nature of the data was highlighted.

## 2.6 Communication Subsystem

In this section, we emphasize the need for a good communication subsystem of computer-aided structural design system. A good



communication link is possible through a well defined languages for interaction between the computer and the designer. Also, computer graphics provides an effective channel of communication.

Languages for interaction are used either through application program or interactively using a computer terminal. Finite element analysis and design optimization application programs interact with the computer to define and manipulate data in the database. Data definition and data manipulation languages are provided for this purpose. These languages are generally a set of commands/subroutine call statements in the host language. It is essential to design these languages such that they are simple and easy to use. In an interactive mode, data definition and manipulation is done using a query language. A general set of interactive commands must be available in the system. Requirements and implementation of these languages are discussed in later chapters.

Finite element analysis and design optimization algorithms produce huge amount of data. In order to make these results useful for interpretation and evaluation, they need to be presented in a readily understandable form. Long list of printed data are not suited for comprehension. Graphical presentations are appropriate solutions. Typical tasks of computer graphics include the selection of visual aids (graphic displays, fast plotters), editing of data to be displayed, command interpretation and graphic database management. Therefore, a well-developed command language and computer-graphics can offer considerable aid to designer to communicate effectively with the computer during the design process.



## 2.7 Users of Computer-Aided Structural Design System

We have to identify different users of computer-aided structural design system and their needs. Three types of users are identified -- (i) the system programmers (ii) application programmers and (iii) interactive users. The difference between these users and the way they use the system are described below.

System programmers are those who develop general purpose programs for structural analysis and optimization. In general, persons who write these programs are not the same as those who apply them. They work on a very high level of data abstraction. They need a good database management system, matrix operation and utility library, and a graphic system. There exists a second category of system programs who modify the existing general purpose finite element programs to add new capability to the programs. Some programs like GIFTS, ADINA, and SPAR have excellent analysis capability. However, many of these systems do not have database management facility capable of sharing data outside the program environment. In some of these programs, a local data management routines are used. These programs can be incorporated into structural design system by providing an interface between the database and the programs or by modification of program to retrieve essential data that program requires. Pre- and post processing capabilities of these programs together with their local database may be used to integrate them in the design process.

The application programmers are much closer to practical applications. Their interest is not to provide means for general problem solution, but to solve special problems; e.g., stress analysis



of a structure for a certain number of load combinations. In general the packaged programs may not have capabilities to handle special needs of the problem. For this reason the application programmers need capabilities to exchange data between subsystems and add their own algorithm whenever the subsystems are not completely covering their needs. Consider, for example a finite element package in which capability to include special boundary conditions does not exist. Application programmer in that case selects an appropriate algorithm for assembly and solution of system of equations to meet the needs. Design optimization procedures have similar needs for selecting alternate application programs. Depending on convergence and other requirements, designer switches to appropriate optimization algorithm, but essentially using almost the same data.

Interactive users are those who use the same application program many times by changing only certain parameters. This type of users do not worry about complicated descriptive or algorithmic facilities. Their concern is data input for many iterations with minimum effort on and easy-to-perceive representation of output. An interactive user of finite element system may like to see the effect of introducing a boundary condition at a particular node or a load at a node. A designer using an optimization package may like to see the convergence pattern for various values of step size increment or the optimization path.

Thus, we have identified the various types of users of a computer-aided structural design system. These are schematically shown in Fig. 2.7.1. Needs of various types of users are considered in developing a computer-aided structural design system.



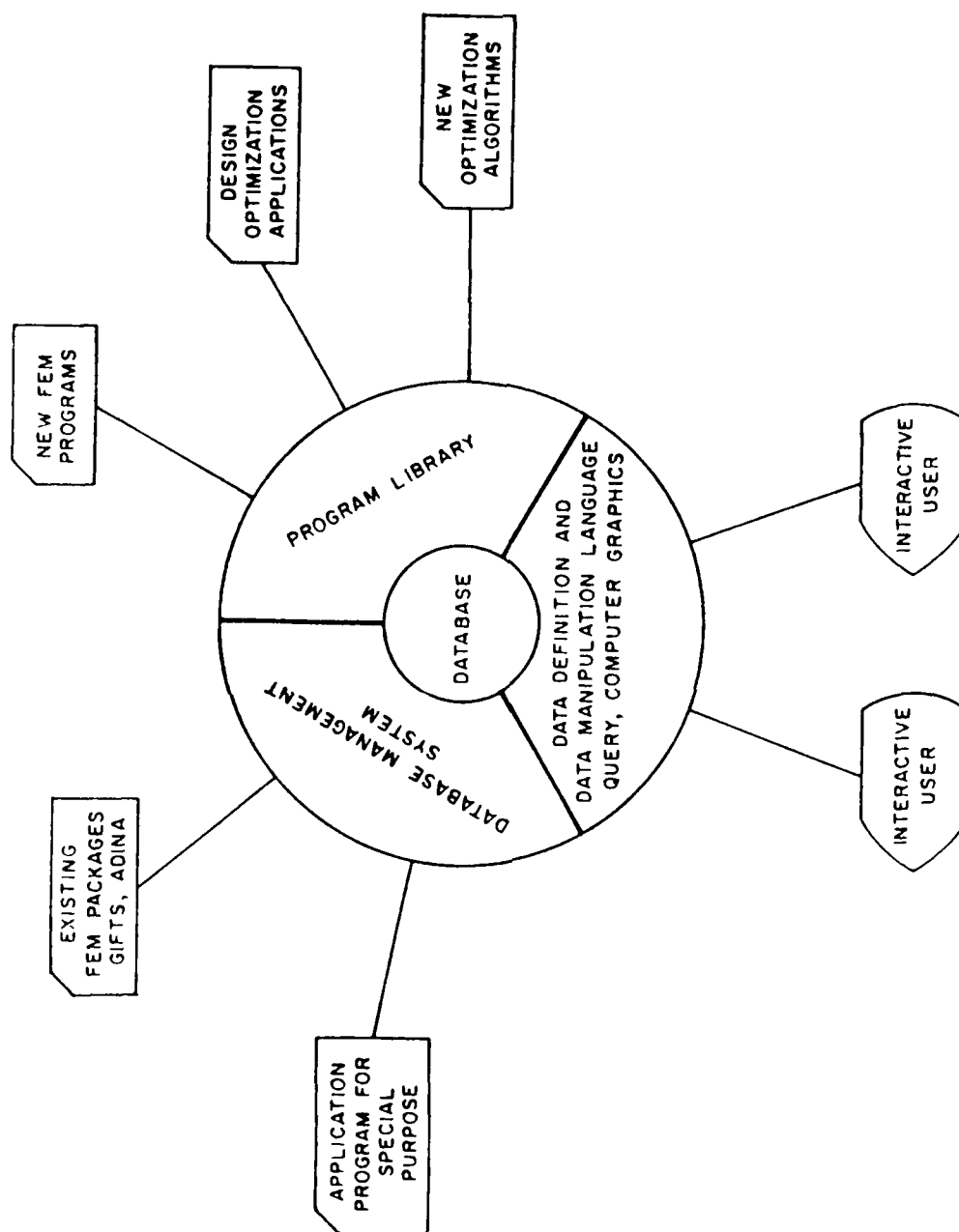


Figure 2.7.1 Users of Computer-Aided Structural Design System



## CHAPTER 3

### A STUDY OF DATABASE MANAGEMENT CONCEPTS

#### 3.1 Introductory Remarks

In the previous chapter, we studied the structural design process and emphasized the importance of database management concepts in computer-aided structural design. One of the objectives of this research is to study various database management concepts. In this chapter this study is made to understand various methods available for data organization and to implement them for structural design applications. The concepts are explained with reference to finite element analysis and design optimization examples. Before the concepts are described problems in developing a good database are posed in Section 3.2. In Section 3.3, various database management terminologies are described, since they are relatively new to engineering community. In Section 3.4, commonly used data models are evaluated in view of organizing structural design data and suitable data models are selected for further study. Concepts of normalization of data is given in Section 3.5. Finally, the concept of global and local databases is explained in Section 3.6. Based on these studies; a technical paper has been recently accepted for publication (Sreekanta Murthy and Arora, 1986).



### 3.2 Problems Associated in Providing a Good DBMS for CAD-OPT

Now, our problem is how the database has to be organized? what kind of information is to be stored? what kind of database management system (DBMS) is suitable? how data is manipulated? and how various finite element analysis and design optimization applications use the data? These problems have to be studied in detail. Data handling techniques in existing finite element analysis programs are primitive and difficult to use. Moreover, they offer little flexibility to extend them for design optimization applications. As new methods of design evolve, there is a need to incorporate the information required by them in the database. Thus it is necessary for the existing database to be flexible and allow simple modifications. The increasing size of database and complexity of information content introduces a new dimension to the problem of inconsistency of data. The operational data creates update consistency problems. If informative data are changed, the operational data must be invalidated. Thus, the problem of data dependency which arises from storing operational data together with informative data influence the design of database.

Abstract structural design information must somehow be modeled into the computer. This modelling aspect of actual design data requires a formal approach. In this regard, sophisticated techniques are available in business database management area to deal with complex data organization problems. But the question arises as to whether engineering data could be similarly modeled? If so, do the structural design databases require different performance consideration from the



database or commercial applications? These questions have not yet been adequately answered. Several different approaches have to be taken; for example, use of commercially available database systems, and development of special structural design database systems. Further, the data modelling considerations depends on the type of user and application programs as discussed earlier. Requirements of users and application programs must be considered for providing a good database organization for structural design.

### 3.3 Definition of Various Terminologies

A number of terminologies and definitions are given to facilitate descriptions in subsequent chapters. They start with simple ones and move on to more complex ones. These terminologies are new to engineering community and are explained here with examples from finite element analysis and structural design applications.

**Database.** A database is defined as a collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple applications. The data are stored so that they are independent of programs which use them. A common and controlled approach is used in adding new data and in modifying and retrieving existing data within the database. The data is structured so as to provide a foundation for future application development. One system is said to contain a collection of databases if they are entirely separate in structure (Martin, 1977).



**Logical Data Structure.** Data in a particular problem consists of a set of elementary items of data. An item usually consists of single element such as integer, real and character or a set of such items. The possible ways in which the data items are structured define different logical data structures. Therefore, it is the data structure as seen by the user of the DBMS without any regard to storage details.

**Model.** The logical structure of data.

**Schema.** The coded form of logical data structure is called schema.

**Data Independence.** It is the property of being able to change the overall logical or physical structure of data without changing the application program's view of the data (Martin 1977).

**Entity.** An entity may be 'anything having reality and distinctness of being in fact or in thought' (Vetter and Maddison, 1981). An entity may be: (i) a real object like structure, material; (ii) an abstract concept like finite element, nodes, a time period; (iii) an event, i.e., a situation that something is happening (e.g., vibrating structure); and (iv) a relationship, e.g., elements of a particular type.

**Entity Set.** An entity set is a collection of entities of the same type that exist at a particular instant, e.g., set of finite elements (ELEMENTS) and set of nodes (NODES).

**Property.** Property is a named characteristic of an entity, e.g., element name, and element material type. Properties allow one to identify, characterize, classify and relate entities.



**Property Value.** It is an occurrence of a property of an entity, e.g., 'element name' has property value BEAM.

**Entity Type.** Entities having same kind of properties are said to be of same type. Upper case letter is used for its name.

**Domain.** A domain is the set of eligible values for a property. A domain has same characteristics as a set, i.e., the values belonging to a domain are distinct and their order is immaterial. A predicate is associated with each domain allowing one to determine whether a given value belongs to the domain in question. Thus, the formal definition of domain  $D_i$  is

$$D_i = \{v_i | P_i\} \quad \text{where } v_i \text{ represents a value satisfying the predicate } P_i$$

Examples of domain are

Element Name = {BEAM, TRUSS, .....}

Element Material type = {STEEL, ALUMINUM, .....}

Length =  $\{x | x > 0 \text{ and } x < 100\}$

**Attributes.** Columns of a two-dimensional table are referred to as attributes. An attribute represents use of a domain within a relation. Attribute names are distinct from those of the underlying domains; e.g.,

Domains:                      NODES =  $\{i | i > 0 \text{ and } i < n\}$   
                                      DOFS =  $\{j | j > 0 \text{ and } j < m\}$



Attributes:            NODE1 - First node of an element derived from  
                          domain NODES  
                          DOF1 - First d.o.f. derived from domain DOFS

Relation:            ELEMENT(E#, NODE1, NODE2)  
                          Attributes NODE1, NODE2 and derived from domain  
                          NODES

**Entity Key.** An entity key is an attribute having different values for each occurring entity and provide unique identification of a tuple. An entity represents a compound key if it corresponds to a group of attributes. It is also called candidate key.

**Primary Key.** If several entity keys exist for a given entity set, then one of them is arbitrarily chosen as the primary key.

**Secondary Key.** It is an attribute that does not have different values for each occurring entity, but identifies those occurring entities that have certain property.

**Relation.**  $R$  is a relation on the domains (i.e., sets)  $D_1, D_2, \dots, D_n$  (not necessarily distinct) if it is a subset of cartesian product  $D_1 \times D_2 \times \dots \times D_n$ . Thus  $R \subseteq D_1 \times D_2 \times \dots \times D_n$ . The value  $n$  represents the degree of the relation  $R$ . The relation  $R$  is usually written as

$$R(D_1, D_2, \dots, D_n)$$

Here  $D_1, D_2, \dots, D_n$  are called attributes of the relations. The values of the attributes are taken from the corresponding domains for



$D_1, D_2, \dots, D_n$ . Note that domain is a set of values where as attribute is a list of values (Vetter and Maddison, 1981).

**Tuple.** Rows of a relations are called tuples.

**Function.** A function is a special kind of relation between two sets, say, A and B. Each member of a set A is associated with exactly one member of set B. A function f is denoted as

$$f: A \rightarrow B$$

**Functional Dependence.** An attribute A is functionally dependent on the attribute B of a relation R if at every occurrence of B-value is associated with no more than one A-value. This is denoted as

$$R.B \rightarrow R.A$$

**Example.** As an example, consider the relation

ELEMENT (ELMT#, EL-NAME, AREA)

EL-NAME is functionally dependent on ELMT#. AREA is functionally dependent on ELMT#. ELMT# is not functionally dependent on EL-NAME, because more than one element could have the same name. Similarly, ELMT# is not functionally dependent on AREA.

An attribute can be functionally dependent on a group of attributes rather than one attribute. For example, consider the relation for a triangular finite element:

CONNECTION (NODE1#, NODE2#, NODE3#, ELMT#)

Here ELMT# is functionally dependent on three nodes NODE1#, NODE2#, and NODE3#. Given any one of NODE1#, NODE2#, or NODE3# it is not



possible to identify ELMT#. These functional dependencies are shown in Fig. 3.3.1.

**Full Functional Dependency.** An attribute or a collection of attributes A of a relation R is said to be fully functionally dependent on another collection of attributes B of R if A is functionally dependent on the whole of B but not on any subset of B. This is written as

$$R.B \twoheadrightarrow R.A$$

In the Fig. 3.3.2 for example, ELMT# in the relation CONNECTION of a triangular finite element is fully functionally dependent on concatenated attributes NODE1#, NODE2# and NODE3# because three nodes combined together define an element. NODE1#, NODE2#, or NODE3# alone does not identify ELMT#.

**Transitive Dependence.** Suppose A, B and C are three distinct attributes or attribute collections of a relation R. Suppose the following dependencies always hold: C is functionally dependent on B and B is functionally dependent on A. Then C is functionally dependent on A. If the inverse mapping is nonsimple (i.e., if A is not functionally dependent on B or B is not functionally dependent on C), then C is said to be transitively dependent on A (refer to Fig. 3.3.3). This is written as

$$R.A \twoheadrightarrow R.B$$

$$R.B \twoheadrightarrow R.A$$

$$R.B \twoheadrightarrow R.C$$





Figure 3.3.1 Functional Dependencies

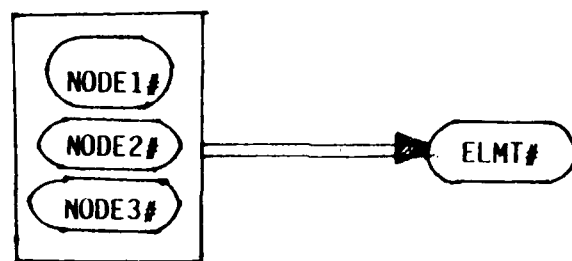


Figure 3.3.2 Full Functional Dependency

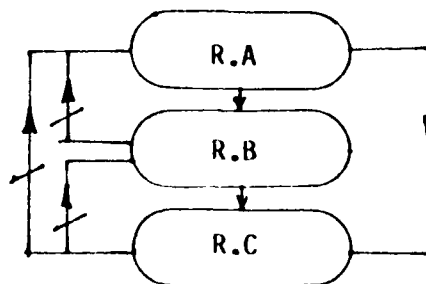


Figure 3.3.3 Transitive Dependence



Then, we can deduce that

$$R.A \rightarrow R.C$$

$$R.C \not\rightarrow R.A$$

For example, consider the relation

$$EL-DISP(ELMT\#, EL-TYPE, DOF/NODE)$$

Here,

$$ELMT\# \rightarrow EL-TYPE$$

$$EL-TYPE \not\rightarrow ELMT\#$$

$$EL-TYPE \rightarrow DOF/NODE$$

Therefore

$$ELMT\# \rightarrow DOF/NODE \text{ (transitively dependent)}$$

$$DOF/NODE \not\rightarrow ELMT\#$$

**Digraph.** A directed graph (refer to Fig. 3.3.4) or a digraph is a figure with nodes and arcs. Each arc is a line with a direction. Nodes represents attributes and arcs represent dependencies (functional, fully functional). Length of a path is the number of arcs in it. For example  $N_1-A_1-N_2-A_3-N_3-A_4-N_2-A_5-N_4$  has length 4. Distance between two nodes is the longest possible path between them. For example distance between  $N_1$  and  $N_4$  is 4 since longest possible path between nodes is  $N_1-A_1-N_2-A_3-N_3-A_4-N_2-A_5-N_4$  (Vetter and Maddison, 1981).

**Connectivity Matrix.** Square matrices can be used to represent digraphs. If the digraph has  $n$  nodes then an  $n \times n$  square matrix is used. Rows and columns represents nodes. Using the value 1 to means presence of a connection and the value 0 to mean absence of a connection, a square matrix can represent the connection between the nodes of a digraph. For example, if node 2 is connected to node 4, then connectivity matrix is



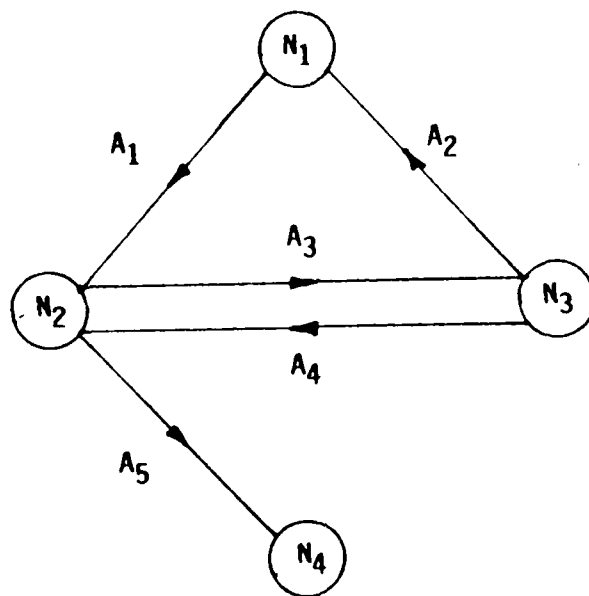


Figure 3.3.4 Digraph



	N <sub>1</sub>	N <sub>2</sub>	N <sub>3</sub>	N <sub>4</sub>
N <sub>1</sub>	0	0	0	0
N <sub>2</sub>	0	0	0	1
N <sub>3</sub>	0	0	0	0
N <sub>4</sub>	0	0	0	0

Each row and column of a connectivity matrix is named the same as the corresponding node (i.e., with the name of the attribute constituting the node). Appropriate role names are then used to distinguish multiple rows and columns denoting essentially a single node (Vetter and Maddison, 1981).

### 3.4 Views of Data Used in Structural Design and Data Models

Nature of data used and computations performed in finite element analysis and structural design optimization were discussed in Sections 2.2 and 2.5. In general, data used in design can be viewed as single data items, vectors, matrices, tables, etc., depending on the nature of data and the context in which they are used. The question of how the design data should be organized in a database can only be answered by considering a formal description of data and associations among them. The most elemental piece of data is a data item. It cannot be further subdivided into smaller data type. A data item by itself is not of much use. It becomes useful only when it is associated with other data items. Thus, database consists of data items and the association among



them. A data model is nothing but a map showing different data items and their associations. A data model shows logical organization of data and is useful to describe user's view of data. Layout of data on physical storage devices is known as physical organization.

A database can be viewed at various levels depending on the context. A level of data representing view of interactive terminal users and application programmers is known as external view. Conceptual view of data deals with inherent nature of data occurring in finite element analysis and design optimization and represents a global view of data which is of theoretical interest. The data organization describing the physical data layout is dealt at the internal level. At this level, one is concerned with efficiency and storage space details. There is one more level of data organization below the internal level where the actual storage of data on a particular computer system becomes the main consideration. But, this aspect is a specialists job and has no general guidelines. Therefore, it is not discussed here. Three levels of data - external, conceptual and internal are used to describe various views of data. These levels of data organization were suggested by ANSI/SPARC (American National Standard Institute/Standard Planning and Requirements committee). They are considered for detailed investigation in the later chapters.

In the following subsections, various methods available for data organization are evaluated in view of organizing structural design data. suitable data models are selected for further study.



### 3.4.1 Hierarchical Model

In this model, the data is represented by a simple tree structure. A tree is composed of a hierarchy of elements called nodes. Every node (except root) has a node related to it at a higher level. The node at a higher level is called a parent node. Each node can have one or more nodes related to it at a lower level called child nodes. A node at the top of a tree is called the root. No node can have more than one parent. A hierarchical model has one-to-many relationships.

In finite element analysis, we can form a hierarchical model with data items such as structure, substructure, and elements. This model is schematically shown in Figs. 3.4.1 and 3.4.2. Another example of hierarchical model is the hypermatrix data organization. Depending on the size of hypermatrix, it is divided into number of submatrices and arranged at various hierarchical levels. In design optimization also, the data related to design variables can be arranged at various hierarchical levels. For example, data of design variable number, design group link, and design variable values can be arranged at various levels. We can see from these examples that hierarchical model fits naturally with the usual subdivision of design data.

### 3.4.2 Network Data Model

In network data model, a child in a data relationship has more than one parent. A network is more general than a hierarchy because a given node occurrence may have any number of immediate superiors as well as



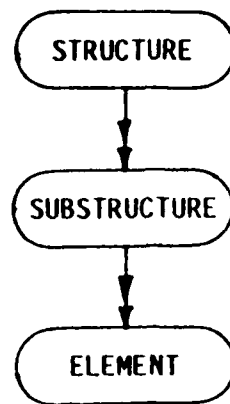


Figure 3.4.1. Hierarchical Data Model

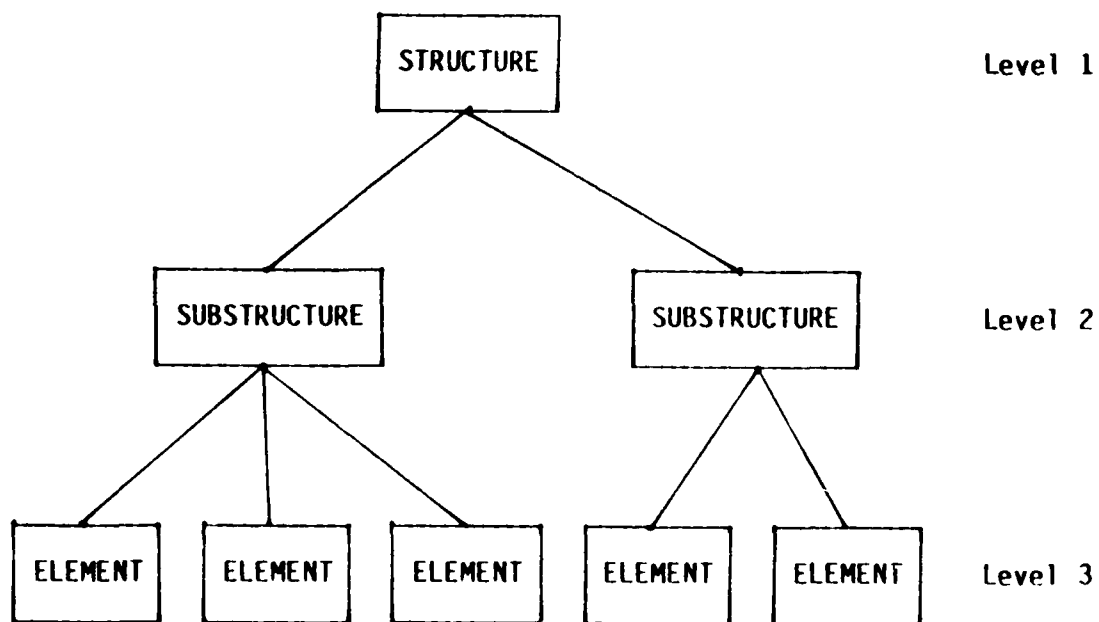


Figure 3.4.2 An Occurrence of a Hierarchical Data Model



any number of immediate dependents. The network model allows many-to-many relationships. A network structure is said to be simple if each directed logical association is functional in at least one direction. This means the model has no line which has double arrows in both direction. Complex network will have double arrows in both directions. Examples of network model for a finite element and node relation is shown in Figs. 3.4.3 and 3.4.4.

### 3.4.3 Relational Model

A data model constructed using relations is referred to as a relational model. A relational data model has a tabular form of data representation. Figure 3.4.5 represent a relational model of data. The rows of the table are referred to as tuples. The columns are referred to as attributes. Relations of degree (defined in Section 3.3) one are said to be unary. Similarly relations of degree two are binary, relations of degree three are ternary, and relations of degree  $n$  are  $n$ -ary. The relational model provides an easy way to represent data and is simpler to use than hierarchical or network data model. The relations can be easily manipulated using special relational operators such as PROJECT, JOIN, etc. The operator PROJECT yields a 'vertical' subset of a given relation, i.e., subset obtained by selecting specified attributes. The operator JOIN puts together columns from different relations. An example of relational model in finite element analysis is node-coordinate relation shown in Fig. 3.4.5.



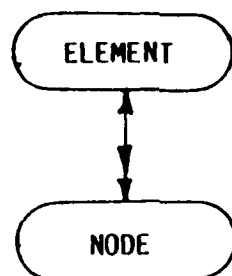


Figure 3.4.3 A Network Model

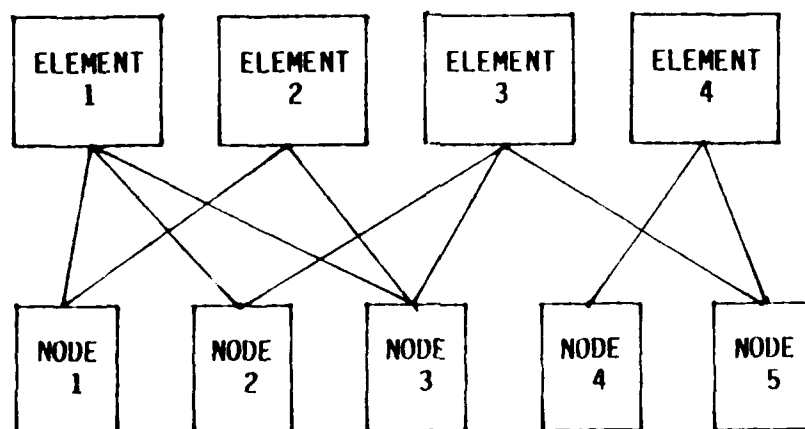


Figure 3.4.4 An Occurrence of a Network Model



CORD

NODE NO	X	Y	Z

Figure 3.4.5 Relational Model

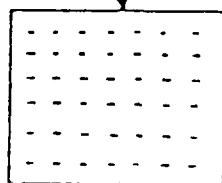


#### 3.4.4 Numerical Model

Most of the computations in design optimization involve operations on matrices like matrix addition, multiplications, solution of simultaneous equations, and eigenvalue calculations. The data models discussed earlier are not tailored to handle matrix data effectively. It is necessary to provide a user-friendly facility for defining such numerical data and manipulating a numerical database. It is possible to provide such a facility by defining a new data model called numerical model (Sreekanta Murthy, Reddy, Arora, 1984). This numerical model is basically a variation of hierarchical data model having two levels of data representation. At the first level information pertaining to type and size of data is placed. The second level contains the actual numerical data. This model is shown in Fig. 3.4.6. A matrix is referenced through a user defined NAME or NUMBER. Various levels of submatrix organization can be defined through parameter called LEVEL. TYPE indicates the type of matrix: square, lower triangular, upper triangular, banded symmetric, banded nonsymmetric, diagonal, etc. ORDER indicates user's view of matrix storage; e.g., rows, columns or submatrices. For storage or retrieval of matrix data, the unit of transaction will be in terms of ORDER; i.e., row, column or full matrix. Dimension of matrix is represented by the number of ROWS or COLUMNS. PRECISION parameter specify the tolerance required while performing floating point operations. NULL parameter specifies if matrix is null or not. By checking this parameter unnecessary operations on null matrices can be eliminated, thereby saving considerable storage and execution time.

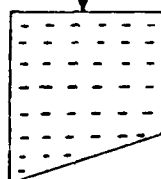


A10	1	SQUARE	ROW	20	20	$10^{-10}$	0
-----	---	--------	-----	----	----	------------	---



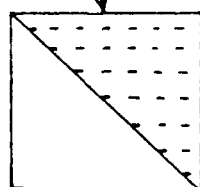
(i) A Square Matrix

B1	0	S-BAND	ROW	20	5	$10^{-10}$	0
----	---	--------	-----	----	---	------------	---



(ii) A Symmetric Banded Matrix

T1	0	U-TRIAN	ROW	15	15	$10^{-10}$	0
----	---	---------	-----	----	----	------------	---



(iii) An Upper Triangular Matrix

Figure 3.4.6 Examples of Numerical Data Model



Examples of numerical model are shown in Fig. 3.4.6. Submatrix organization for level 1 is shown in Fig. 3.4.6(i). The matrix information for level 0 can be described in the same way as shown for level 1. Symmetric banded matrix organization is shown in Fig. 3.4.6(ii). Note that rows or columns are of variable size. Figure 3.4.6(iii) shows an upper triangular matrix represented in the model.

### 3.4.5 Choice of Data Model for Structural Design

It was seen that various types of data models described in the previous sections could be used for structural analysis and design optimization applications. However, it is not possible to expect effective use of any one of the models in all situations. Hierarchical data model is clearly superior in organizing general engineering data which occur naturally in hierarchical form. For example, large order matrices can be assembled using submatrices. These submatrices can be organized at various hierarchical levels. Network model handles more general situation than hierarchical model. The disadvantage of this model is the complexity associated with its use. Both hierarchical and network data models use a fixed structure and offer little flexibility to change for alternative structure. Another drawback of the hierarchical model is the complexity of database design requiring tedious process of establishing links between data. If new kinds of data are to be added or new information must be generated from a database, it is necessary to add new links. Generally, this process requires redesigning the entire database. However, a relational data



model uses a less preconceived structure and provides user-friendly data representation. This model can provide easy access to data for the user. Also, the tabular structure of the model provides a convenient way of representing structural design data that are generally in the tabular form. A major advantage of this model is the ease with which database can be changed. As the design evolves, new attributes and relations can be added, and existing ones deleted easily. It is possible to support simple query structure in the relational model. In situations where application programs require a complete set of related items together, retrieving parts of information is not useful. In such a case the relational model which is set oriented provides a suitable way to organize the design data. Numerical model appears to be quite effective in representing matrix data structure.

Relational and numerical data models are therefore selected for detailed study for design of the database and the development of database management system for structural analysis and optimization applications.

### 3.5 Normalization of Data

It was seen in the previous section that data items are grouped together to form associations. An issue of concern here, is how to decide what data items have to be grouped together? In particular, using a relational model, determining what relations are needed and what their attributes should be? It was emphasized in Chapter 2 that structural design data gets constantly modified, updated and deleted.



As database is changed, older views of data must be preserved so as to avoid having to rewrite the programs using the data. However, certain changes in data associations could force modification of programs, and could be extremely disruptive. If grouping of data items and keys is well thought out originally, such disruptions are less likely to occur.

Normalization theory (Date 1982) provides certain guidelines to organize data items together to form relations. The theory is built around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. Three normal forms - 1st, 2nd and 3rd - are described below.

**First Normal Form (1NF).** A relation is said to be in first normal form if and only if it satisfies the constraint of having atomic values.

As an example, Fig. 3.5.1 shows the relation CONN between four attributes ELMT#, E-NAME, NODES#, and DOF/NODE with domains  $D_1$ ,  $D_2$ ,  $D_3$  and  $D_4$ . The relation is first shown not in 1NF and then it is shown in the 1NF.

**Second Normal Form (2NF).** A relation is in the second normal form if and only if it is in 1NF and every non-key attribute is fully functionally dependent on each candidate key.

Let us see if the relation CONN of Fig. 3.5.1 in the 1NF is also in 2NF. Consider a non-key attribute E-NAME:

ELMT#, NODES#  $\rightarrow$  E-NAME

ELMT#  $\rightarrow$  E-NAME

NODE#  $\nrightarrow$  E-NAME



Domain  $D_1$ 

ELMT#1
ELMT#2
ELMT#3

Domain  $D_2$ 

BEAM
TRUSS
PLATE

Domain  $D_3$ 

NODE#1
NODE#2
NODE#3
NODE#4
NODE#5

Domain  $D_4$ 

No. of DOF per node
6
3
2

CONN	Key		Key	
	ELMT#	E-NAME	NODES #	DOF/NODE
	1	BEAM	1 2	6
	2	TRUSS	3 5	3
	3	PLATE	2 3 4 5	2

Not in 1NF

Key		Key	
ELMT#	E-NAME	NODES #	DOF/NODE
1	BEAM	1	6
1	BEAM	2	6
2	TRUSS	3	3
2	TRUSS	5	3
3	PLATE	2	2
3	PLATE	3	2
3	PLATE	4	2
3	PLATE	5	2

In 1NF

Figure 3.5.1 First Normal Form for a Relation CONN



Therefore,  $ELMT\#, NODES\# \not\rightarrow E-NAME$ , i.e.,  $E-NAME$  is not fully functionally dependent on  $(ELMT\#, NODES\#)$ .

Similarly for the non-key attribute  $DOF/NODE$ :

$ELMT\#, NODES\# \rightarrow DOF/NODE$

$ELMT\# \rightarrow DOF/NODE$

$NODE\# \nrightarrow DOF/NODE$

Therefore,  $ELMT\#, NODES\# \not\rightarrow DOF/NODE$ . Since neither  $E-NAME$  nor  $DOF/NODE$  are fully functionally dependent on candidate key  $(ELMT\#, NODES\#)$  the relation  $CONN$  is not in 2NF.

Conversion of the relation  $CONN$  to 2NF consist of replacing  $CONN$  by two of its projections (refer to Fig. 3.5.2).

$NAM-DOF \leftarrow CONN (ELMT\#, E-NAME, NODES\#, DOF/NODE)$

$ELMT-NODE \leftarrow CONN (ELMT\#, E-NAME, NODES\#, DOF/NODE)$

Relation  $ELMT-NODE$  does not violate 2NF because its attributes are all keys.

**Third Normal Form (3NF).** A relation is in third normal form if it is in second normal form and every non-prime attribute is non-transitively dependent on each candidate key of the relation.

For example, consider the relation  $NAM-DOF$  (Fig. 3.5.2) to see if it is in third normal form. It still suffers from a lack of mutual independence among its non-key attributes. The dependency of  $DOF/NODE$  on  $ELMT\#$ , though it is functional, is transitive (via  $E-NAME$ ). Each  $ELMT\#$  value determines an  $E-NAME$  value and in turn determines the  $DOF/NODE$  value. This relation is reduced further into relations  $NAME$  and  $DOF$ . These relations (Fig. 3.5.3) are in third normal form.



NAM-DOF	ELMT#	E-NAME	DOF/NODE
	1	BEAM	6
	2	TRUSS	3
	3	PLATE	2

ELMT-NODE	ELMT#	NODE#
	1	1
	1	2
	2	3
	2	5
	3	2
	3	3
	3	4
	3	5

Figure 3.5.2 Second Normal Form for Relation CONN

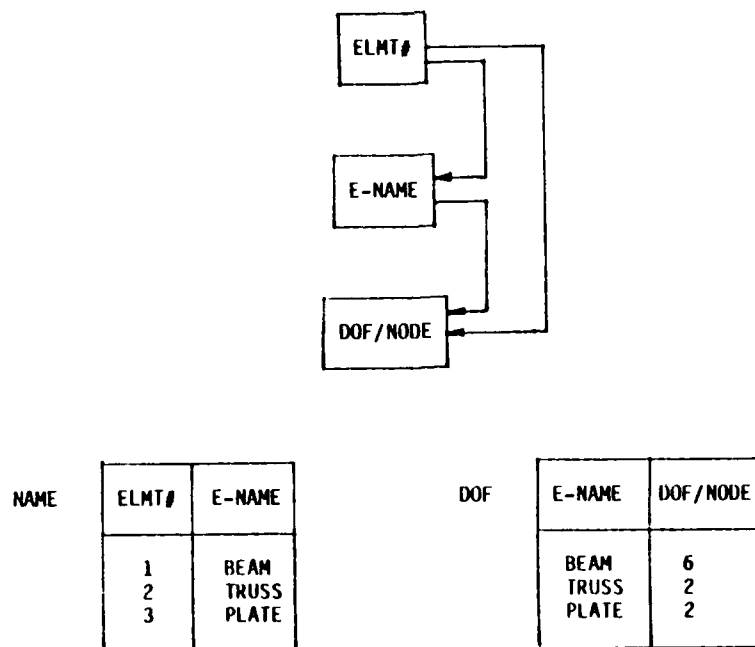


Figure 3.5.3 Third Normal Form for Relation NAM-DOF



From the above examples, we see that the concept of normalization of data provides a good basis to group different data items of structural design. Therefore, the concept of normal forms of data are used in this study to develop a methodology to design a database for structural design. Drawback in using this concept is that its use requires a rigorous analysis of data dependencies and their associations. This analysis of data looks complicated at this stage of research on database management for structural design optimization. Also, these normal forms of data do not suggest how large matrix data should be organized in a database. However, normalization concept is useful in organizing general data used in finite element analysis and design optimization.

### 3.6 Global and Local Databases

Computer-aided design of complex structural systems uses several application programs during the design process. Many of these programs require common information such as geometry of the structure, finite element idealization details, material properties, loading conditions, structural stiffness, mass and load distributions, and responses resulting from analysis runs. Also, it is common that data generated by one program is required for processing in subsequent programs in certain predetermined pattern. These data do not include transitory information such as intermediate results generated during an analysis run. The transitory information is highly unstructured and its usage pattern is known only to applications that use them. Generally, the transitory



information is deleted at the end of a run. Therefore, there is a need for systematic grouping of the data.

A network of databases offers a systematic approach to satisfy the need stated above. A network of databases consists of a global database connected to a number of local databases through program data interface. Application programs which use them may be thought as links connecting the databases (Fig. 3.6.1). A global database contains common information required for all applications whereas a local database contains only application dependent transitory data. Data in global database is highly structured and integrity of the database is maintained carefully. However, data in local database is extremely flexible and integrity is not of importance.

This network of databases offers considerable aid in structural design process. Any changes made to the data in global database is immediately available for use in other applications of the system. Any new application program can be added to share the common data. The data views in global database are clear to all applications and any modified views can be easily incorporated to suit a new application. Local databases are dependent on application programs and are highly efficient in data transfer operations since no overhead is involved in maintaining complicated data structures. It supports trial and error design process by providing scratch pad work space which can be erased from the local database at a specified design stage. Any intermediate results can be stored in a local database. Summarized and final results of design can be transferred to a global database.



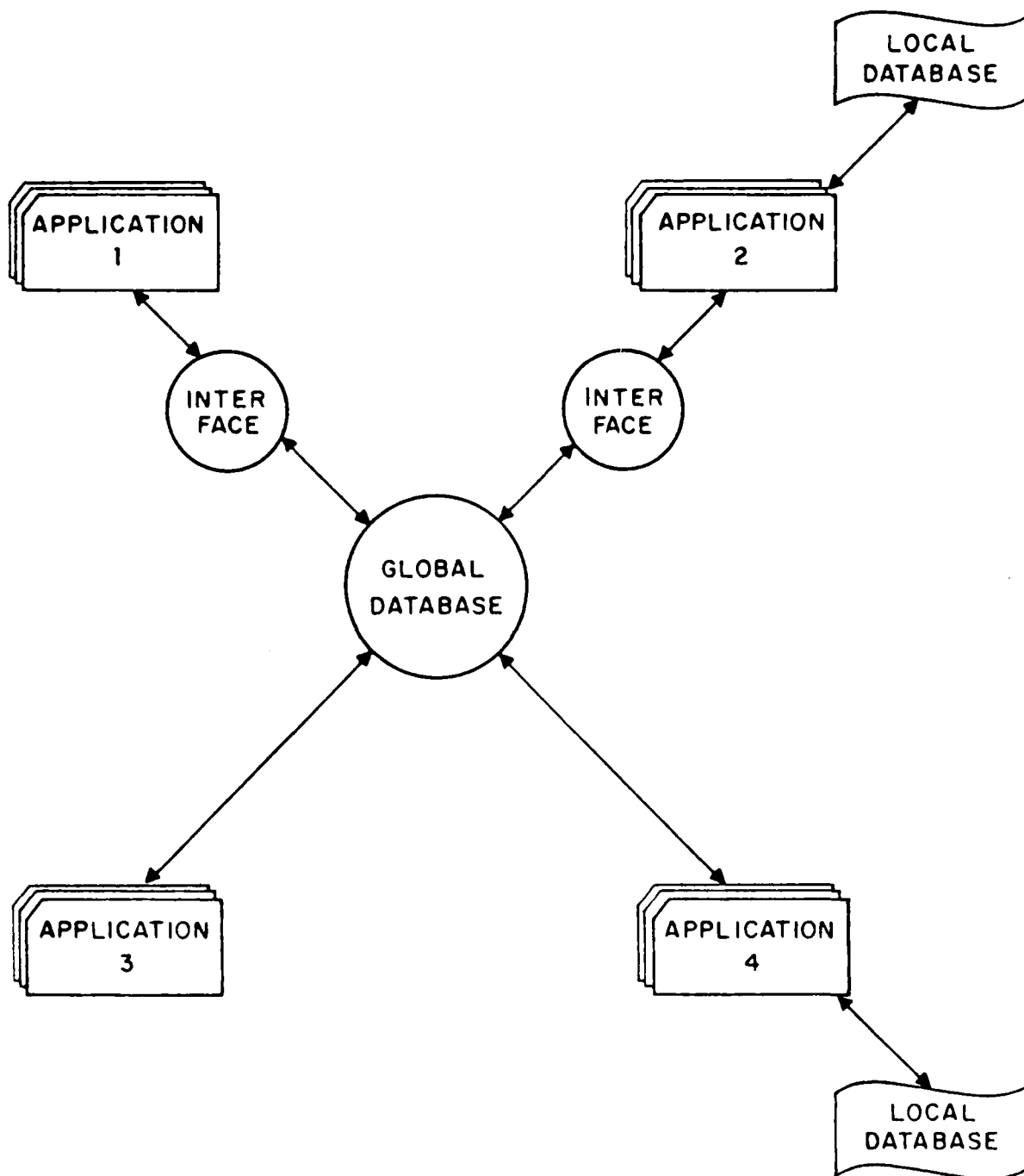


Figure 3.6.1 Network of Databases



## CHAPTER 4

### DATABASE DESIGN METHODOLOGY FOR STRUCTURAL ANALYSIS AND DESIGN

#### 4.1 Introductory Remarks

A methodology for designing databases for structural design is proposed in this chapter. The methodology is based on the database management concepts described in the previous chapter. Background for developing the database design methodology is given in Section 4.2. In the proposed methodology relational and numerical data models are used. Three levels of data organization -- conceptual, internal and external are suggested for structural design database. In Section 4.3, a methodology for constructing a conceptual data model is described. The conceptual model enables us to find out the inherent nature of structural design data irrespective of computer program constraints. Since large amount of data storage and speed of accessing data are required in finite element analysis and optimization, we need to consider efficiency of storage space and I/O. This aspect is considered in Section 4.4. A methodology for developing an internal model is given there. In Section 4.5, a methodology for developing an external data model is described. External model provides data needed for multiple users or application programs according to their individual perspectives of data. Matrix data constitutes a large portion of finite element analysis and optimization data. Such data needs special considerations



for accommodation in a database. A methodology for organizing large matrix data is given in Section 4.6. Finally in Section 4.7 algorithmic modelling is given. Based on this work, a paper has been recently submitted for publication (Sreekanta Murthy and Arora, 1985b).

#### **4.2 Aspects Considered in the Proposed Methodology and Background**

Several methodologies for designing databases for business applications have been researched in the past few years. However, in engineering database management area, there does not exist any rigorous study on design of a database. The existing finite element analysis and design optimization programs use primitive data organization techniques for their out-of-core data storage needs. The drawbacks of such schemes were pointed out earlier.

The question is -- is it possible to develop a methodology for designing a database for structural analysis and design? If possible, how do we begin to develop a methodology? Are the methodologies used in business application suitable for our purpose? If not, what aspects should be considered in developing a methodology for structural design databases? All these questions have to be considered to arrive at a methodology for database design needs.

At this stage, we know that it is possible to identify and group the data used in finite element analysis and design optimization. We also concluded that relational and numerical data models are suitable for organizing the data. The basic problem is that once all the data items have been identified, how to combine them to form useful



relations. In business applications database design follows some well defined steps. First step is the extraction of all the characteristics of the information which is to be represented in the database. Analysis of the information and their integration into one conceptual model is the second step. The conceptual data model obtained by this process is abstract. It is independent of any computer restraint or database management software support. In order for the conceptual model to be useful, it must be expressed in terms compatible with a particular DBMS by considering efficiency of storage space and access time. An internal model is developed for this purpose which is compatible with the conceptual data model. Finally, the database design requires accommodation of different users of the database by providing an external data model. The systematic process by which one traverses the different steps of database design and performs the mapping from one level of abstraction to the next is called a database design methodology.

Suitability of some of the methodologies used in design of databases for business applications to design databases of computer-aided design applications has been investigated by Buchman and Dale (1979). The investigators analyzed three existing methodologies -- Bubenko's methodology, Kahn's methodology, and Smith and Smith methodology with reference to applications in engineering design of a chemical plant. A list of criteria for evaluating the methodologies is given. Salient characteristics of these methodologies are outlined here. In Bubenko's methodology, entities are identified and classified



from query and transaction descriptions. A strong point of this method is the provision for two levels of abstractions. Grouping of entities, however, is highly intuitive and application dependent. Kahn's approach has characteristics of separating the database design problem into two perspectives: information structure perspective which describes the interconnection of information, and the usage perspective which deals with processing requirement of information. The method requires design of two schema - one processing and the other information oriented - which are merged at the end of database design to get a conceptual model. This methodology merges local views into the global view by aggregating entities. Designers intuition is required in this methodology to form nonredundant entities and relations. Smith's methodology ignores information analysis and considers only abstract objects of interest. An object can be viewed as entity, attribute, or relation depending solely on the view point of the user. The abstraction step used in this method is highly intuitive. Grabowski and Eigner (1979) pointed out the necessity for a semantic model construction in CAD application. They described three available semantic models using the example of a geometric model of a line: (i) based on binary association, (ii) based on entity and attribute association, and (iii) expanded relational model.

The methodologies described above are at the research state in business data management field and are not suitable for engineering applications. Also many of them do not discuss details of the procedure or implementation aspect and hence cannot be directly used for designing



an engineering database. Therefore, it is necessary to adopt good features and guidelines provided by existing methodologies and arrive at a suitable one to design databases for finite element analysis and structural design optimization applications.

The proposed methodology to design databases (Sreekanta Murthy and Arora, 1985b) considers several features and requirements of finite element analysis and structural design optimization applications. Some important features of data considered in the methodology are -- tabular structure, matrices, static information, operational information, multiple views of data for different application, and iterative changes in data. Tabular structure of data is organized using relational data model, whereas large matrix data needs is organized using a numerical data model. A simpler approach to design a database is by considering static and operational information separately for an initial design and later merging them to arrive at a final design. Multiple views of data necessary to accommodate theoretical, implementational and user's requirements are considered. Conceptual, internal and external views of data suggested by ANSI/SPARC is considered to accommodate multiple views of data. The methodology uses entity set, relationship set and attributes to form syntactic basic elements of the conceptual model.

In summary the methodology developed here considers the following aspects: (i) Three views of data -- conceptual, internal, and external as suggested by ANSI/SPARC; (ii) Entity set, relationship set, and attributes to form syntactic basic elements of the conceptual model; (iii) Relational data model; (iv) Matrix data; (v) Processing requirements; and (vi) Normalization of data for relational model.



### **4.3 Methodology to Develop a Conceptual Data Model**

#### **4.3.1 Basic Considerations**

Our objective now is to develop a methodology to form a conceptual data model at a suitable level of abstraction regardless of whether or not the available database management software supports such model directly. The conceptual data model should serve as a central reference point for all applications using the database. It changes only if changes in the structural design process occur. Any change in the database management software or application program should not affect the conceptual data model. The model should be capable of supporting new applications with the existing types of data as well as incorporating further data types as needed. Therefore, an analysis of data used in structural design is required to incorporate the features of data model described above. In the analysis, the information in use or needed in future is identified, classified and documented. This forms a basis for the conceptual data model to represent structural design data and design process as a whole.

Two basic approaches are proposed to develop a methodology to form a conceptual data model. One approach considers the general data of finite element analysis and design optimization such as geometry, material properties, element connectivity, element level vectors and matrices. A conceptual model is developed for these types of data. Since large matrices such as assembled stiffness, load, and mass matrices have basically different characteristics a separate approach is proposed in the later sections of this chapter to construct a conceptual



data model. The two approaches together form a methodology to develop a conceptual data model for both types of data used in finite element analysis and design optimization.

The following steps are proposed to develop a conceptual data model for general data. These steps are discussed in detail in Subsections 4.3.2 to 4.3.5.

**Step 1.** Identify all the characteristics of data used in structural analysis and design optimization.

**Step 2.** Data identified are stored in a number of relations. They are reduced to elementary relation which represent inherent association of data.

**Step 3.** More elementary relations are derived from the ones formed in Step 2. This step uncovers more relationships between basic data collected in Step 2.

**Step 4.** Redundant and meaningless relations obtained in Step 3 are removed to obtain conceptual data model.

The conceptual model obtained by this process is abstract and is of theoretical interest representing inherent nature of structural design data and is independent of any computer restraint or database management software support.

#### **4.3.2 Identification of Conceptual Data Objects**

The following steps are proposed to identify the conceptual data objects used in structural design:



BD-A174 450

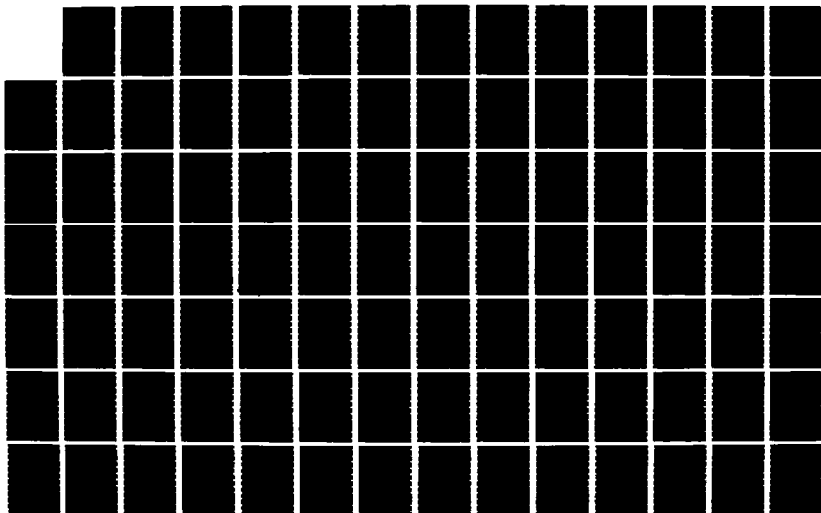
COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION USING A  
DATABASE MANAGEMENT (U) IOWA UNIV IOWA CITY OPTIMAL  
DESIGN LAB T SREEKANTAMURTHY ET AL 30 SEP 86

2/4

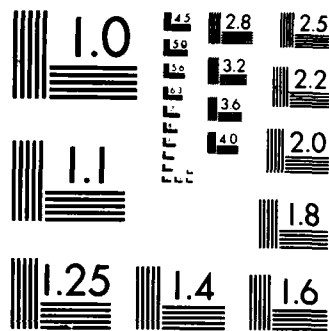
UNCLASSIFIED

ODL-85-17 AFOSR-TR-86-2069 AFFSR-82-0322 F/G 18/3

NL







MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A



**Step 1.** Identify each type of entity and assign a unique name to it.

**Step 2.** Determine the domains and assign unique names to them. This step identifies the information which will appear in the model, such as attributes.

**Step 3.** Identify the primary key for each type of entity depending on the meaning and use.

**Step 4.** Replace each entity set by its primary key domains. Determine and name relations corresponding to association between primary key domain and other domains. Also, include relations corresponding to association between entities themselves. This step gives a collection of relations forming a rough conceptual data model.

Modified definition of domain and attribute are suggested as follows to suit correct identification of structural design information.

**Domain.** A domain is the set of eligible values for a property. A domain has same characteristics as a set, i.e., the values belonging to a domain are distinct and their order is immaterial. A domain can contain vectors or matrices. Thus domain  $D_i$  is defined as

$$D_i = \{v_i | P_i\}$$

where  $v_i$  represents a value, a vector, or a matrix satisfying the predicate  $P_i$ .



**Attribute.** Columns of a two-dimensional table are referred to as attributes. An attribute value can contain relation names and null values.

**Example.** We consider the sample structural design problem given in Section 2.2, to describe these steps.

**Step 1.** The following entity sets are identified for the structure:

STRUCTURE	(S)
BEAM	(B)
TRUSS	(T)
MEMBRANE-TRI	(TR)
MEMBRANE-QD	(QD)
NODE	(N)
ELEMENT	(E)

**Step 2.** We identify the following domains:

STRUCTURE#	Structure identification number (integer)
B#	Beam element identification number (integer)
T#	Truss element identification number (integer)
TR#	Triangular membrane element identification number (integer)
QD#	Quadrilateral membrane element identification number (integer)
NODE#	Node number (integer)
E#	Element number (integer)



EL-TYPE	Element type {BEM2, BEM3, TRS2, TRS3, TRM2, TRM3, QDM2, QDM3} (character)
MATID	Material identification code, e.g., {STEEL•1, STEEL•2, ALUM•5, COMP•1}. It also refers to relation or table of material properties; for example, STEEL•1 refers to relation STEEL and material subtype 1 (character)
MATPRO	Material property {E, $\mu$ , G, ...} (real)
CSID	Cross-section type identification code; e.g., THICK•1, THICK•2, RECT•1, CIRC•5, ISEC•6, LSEC•15}. It also refers to a relation of cross-sectional details. For example, RECT•1, refers to a relation RECT and cross-section subtype 1 (character)
CSPRO	Cross-sectional property {H, W, T, R, ...} (real)
DOF#	Degrees of freedom numbers (integer)
LOAD-TYP	Load type {CONCENTRATED, DISTRIBUTED, TEMPERATURE, ACCELERATION} (characters)
X	X coordinate (real)
Y	Y coordinate (real)
Z	Z coordinate (real)
DESCRIPTION	Description (characters)
VEC	Vectors {Integer, Real and Double precision vectors} (vector)



MATX	Matrices {Integer, Real and Double precision matrices} (matrices)
VECID	Vecto identification code (character) $\equiv \{x \cdot y \mid x = \text{vector description, } y = \text{number}\}$ ; e.g., FORCE•5, LOAD•10
MAXID	Matrix identification code (character) $\equiv \{x \cdot y \mid x = \text{matrix description, } y = \text{number}\}$ ; e.g., EL-STIFF•10, EL-MASS•5

**Step 3.** The following entity keys are identified

STRUCTURE#	for entity set structure
BEAM#	for entity set beam
TRUSS#	for entity set truss
TR#	for entity set TR
QD#	for entity set QD
E#	for entity set element

**Step 4.** In the association between entity sets and domain, the entity sets from Step 1 are replaced by their primary keys. Attribute names are derived from domain names to provide role identification. The following relations are identified:

For Entity Set STRUCTURE

STRUCTURE (S#, DESCRIPTION, MAXID, MATX)

The structure is identified by a structure number S#. Name of the structure and other details are given in DESCRIPTION. Matrices associated with the structure are identified by MAXID.

For Entity Set BEAM



BEAM (B#, E#, EL-TYP, MATID, E,  $\mu$ , G, NODE1#, NODE2#, CSID,  
H, W, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

A beam is identified by a beam number B#. Element number E# uniquely identifies the finite elements of a structure. Attributes NODE1# and NODE2# are derived from domain of node numbers. Similarly, E,  $\mu$ , and G are role names for domain of material property values. CSID identifies the cross-section properties H, and W. Vectors and matrices associated with the element are identified through VECID and MAXID, respectively.

Similarly the relations TRUSS, TRM, and QDM are as follows:

TRUSS (T#, E#, EL-TYPE, MATID, E, NODE1#, NODE2#, CSID, H, W  
LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

TRM (TR#, E#, EL-TYPE, MATID, E, NODE1#, NODE2#, NODE3#,  
CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

QDM (QD#, E#, EL-TYP, MATID, E, NODE1#, NODE2#, NODE3#,  
NODE4#, CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID,  
MATX)

NODE (NODE#, X, Y, Z, DOF1#, DOF2#, DOF3#, LOAD-TYP, LOAD#,  
VICID, VEC)

ELEMENT (E#, NODE#)

From the above example, the following points unique to structural design databases should be noted:

1. The attributes in the example contain relation names (e.g., MATID, CSID). These relations are again association between an entity set; e.g., STEEL and domain of material properties.



2. Null values of domains (which are attributes in the relations) are allowed. For example, in relation TRM, LOAD-TYP and LOAD# may be null if no loads exists.
3. A row and a column intersection may not be a single value; it can be a vector or a matrix.

#### 4.3.3 Reduction to Elementary Relations

In the previous section, we described a method to identify entities, domains and relations to produce a rough conceptual model of data used in finite element analysis and design optimization. Our idea is to develop a conceptual model which contains all the facts and each fact occurring only once. For this purpose, we suggest the use of concept of elementary relations (Vetter and Maddison, 1981) to transform the rough conceptual data model into a better model.

A relation is irreducible if it cannot be broken down by means of project operations into several relations of smaller degree such that these relations can be joined to reconstitute the original relation. A relation which is not reducible is called an elementary relation.

To see how elementary relations satisfy the requirement that one fact is recorded in one place, an example is given below:

LOAD (N#, LC#,  $F_x$ )

where N# is node number

LC# is load case number

$F_x$  is force in the x-direction



LOAD

N#	LC#	F <sub>x</sub>
N1	L1	10.0
N1	L2	15.0
N1	L3	10.0
N2	L2	20.0
N2	L3	20.0
N2	L4	10.0
N3	L3	20.0

Note that this relation describes two facts--load cases, and load values for each node. Now, to see if this is an elementary relation, suppose we split the relation LOAD by means of project operations into following two relations R1 and R2:

R1(N#, LC#)

N#	LC#
N1	L1
N1	L2
N1	L3
N2	L2
N2	L3
N2	L4
N3	L3

R2(LC#, F<sub>x</sub>)

LC#	F <sub>x</sub>
L1	10.0
L2	15.0
L2	20.0
L3	10.0
L3	20.0
L4	10.0

On joining R1 and R2, R1\*R2 we get



R3

N#	LC#	F <sub>x</sub>
N1	L1	10.0
N1	L2	15.0
→ N1	L2	20.0
N1	L3	10.0
→ N1	L3	20.0
→ N2	L2	15.0
N2	L3	10.0
→ N2	L3	20.0
N2	L4	10.0
→ N3	L3	10.0
N3	L3	20.0

← This value comes from  
<N1,L2> and <L2,20.0>

The rows marked with → are not in the original relation LOAD and hence not correct. Thus, the relation is irreducible, and it is an elementary relation. Observe that in the relation LOAD the attribute F<sub>x</sub> is fully functionally dependent on N# and LC#. N# alone or LC# alone does not determine F<sub>x</sub>. Therefore, it is possible to identify such dependencies and establish rules for reducing a relation to an elementary relation. Using the concept of functional dependencies, full functional dependencies, and transitive dependencies, the following steps are identified to form elementary relations:

**Step 1.** Replace the original relations by other new relations to eliminate any (nonfull) functional dependencies on candidate keys.

**Step 2.** Replace the relations obtained in Step 1 by other relations to eliminate any transitive dependencies on candidate keys.



**Step 3.** Go to Step 5 if

- (a) relation obtained is all keys, and
- (b) relation contains a single attribute that is fully functionally dependent on a single candidate key.

**Step 4.** Determine primary key for each relation which may be a single or composite attribute. Take projections of these relations such that each one contains a primary and a non-primary key.

**Step 5.** Elementary relations obtained.

**Example.** We can see how these steps are applicable to relations of the example problem in the previous section. Consider the relation TRM:

TRM (TR#, E#, EL-TYPE, MATID, E, NODE1#, NODE2#, NODE3#, CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

<u>Dependencies</u>	<u>Remarks</u>
TR#	Primary key
TR# → E# also E# → TR#	Secondary key E#
TR# → EL-TYPE	Element type is functionally dependent on TR#
TR# → MATID	Material identification code MATID is functionally dependent on TR#
MATID → E	Material property E is functionally dependent on MATID
TR# → MATID → E	Material property E is transitively dependent on TR# through MATID



TR# identifies NODE1#

TR# → NODE1#

TR# → NODE2#

TR# → NODE3#

TR# → CS-TYP → T

Thickness is transitively dependent  
on TR# through CS-TYP

TR# → LOAD-TYP

TR# → LOAD#

TR# → VECID → VEC

Any vectors, such as load, force,  
stress vector are transitively  
dependent on TR# via VECID

TR# → MAXID → MATX

Similarly matrices such as stiffness,  
mass, are transitively dependent on TR#  
via MAXID

Reducing relation TRM to elementary relations

### Step 1.

ER1 (TR#, E#)

ER2 (TR#, EL-TYP)

ER3 (TR#, NODE1#)

ER4 (TR#, NODE2#)

ER5 (TR#, NODE3#)

ER14 (E#, TR#)



**Step 2.**

ER6 (TR#, MATID);	ER7 (MATID, E)
ER8 (TR#, CSID);	ER9 (CSID, T)
ER10 (TR#, VECID);	ER11 (VECID, VEC)
ER12 (TR#, MAXID);	ER13 (MAXID, MATX)

**Step. 3** The above relations contain single attribute, so go to Step 5.

**Step. 4.** Skip**Step 5.** ER1 to ER13 are elementary relations.

These steps are used on the rest of the relations identified earlier to get a set of elementary relations for the sample structural problem.

**4.3.4 Determination of Transitive Closure**

We generally obtain hundreds of elementary relations when the steps given above are applied to actual finite element analysis and design optimization data. While deriving a large number of relations for obtaining a conceptual data model, it is possible that some relations might have been missed. In general, it is possible to derive further elementary relations from any incomplete collection of such relations. To explain in a simple way, how such additional relations can be derived, consider two relations ER1(A,B) and ER2(B,C), which imply functional dependencies:  $A \rightarrow B$  and  $B \rightarrow C$ . We know that product of functional dependencies leads to transitive dependencies (Vetter and

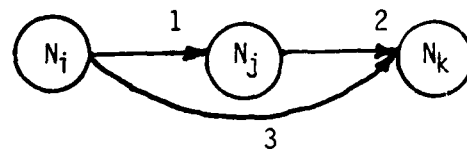


Maddison, 1981). Taking product of above functional dependencies, we get  $A \rightarrow C$ . Therefore, from suitable pairs of elementary relations representing functional dependencies further elementary relations can be derived. Deriving all such relations from initial collection of elementary relations yields a transitively closed collection of elementary relations called transitive closure (Vetter and Maddison, 1981). This set of relations includes both derived and original elementary relations. It is complete in the sense that all elementary relations equivalent to transitive dependencies through others are included. There is always a unique transitive closure for a given set of elementary relations. A transitive closure usually contains many redundant relations. We say that an elementary relation is redundant if it is derivable from other relations.

There are problems associated in interpreting relations in transitive closure. For example, consider relations ER1 (TR#, MATID) where  $TR\# \rightarrow MATID$  and ER7 (MATID, E) where  $MATID \rightarrow E$ . Transitive closure for this set yields relation ER (TR#, E) implies TR# identifies E. This relation does not, however, represent true information as material property E is dependent only on material number and not on element number. The relation could be wrongly interpreted. Therefore such semantically meaningless dependencies must be eliminated.

It is possible to determine transitive closure using directed graphs and the connectivity matrix (defined in Section 3.2). The nodes of graph correspond to entity keys and arcs correspond to elementary relations.





Transitive closure is formed by adding arc 3 if arcs 1 and 2 already exist. A diagram for the elementary relations formed in the previous section is shown in Fig. 4.3.1.

A connectivity matrix for this diagram is shown in Fig. 4.3.2. In the connectivity matrix, we can see for example,  $TR\# \rightarrow MATID$  is indicated by 1 in the corresponding row(3) and column(7) of the matrix. Derived transitive dependence of example  $TR\# \rightarrow E$  is recorded by assigning 1 to  $C(3,11)$ . Such derived relations are denoted by  $1^*$  in the Fig. 4.3.2 and are denoted by new arcs in dotted lines of Fig. 4.3.1. An algorithm for determining transitive closure is given in Appendix I.

The transitive closure for the example produces additional dependencies given in Table 4.3.1. We have eliminated meaningless dependencies from the list.

#### 4.3.5 Selecting Elementary Relations to Form a Conceptual Data Model

In the previous section, we derived additional elementary relations from a set of original elementary relations of Section 4.3.3. Now, we have all the data items and their associations in terms of a large number of elementary relations. We will see that many relations are redundant and therefore they should be removed to provide a minimal set of elementary relations.



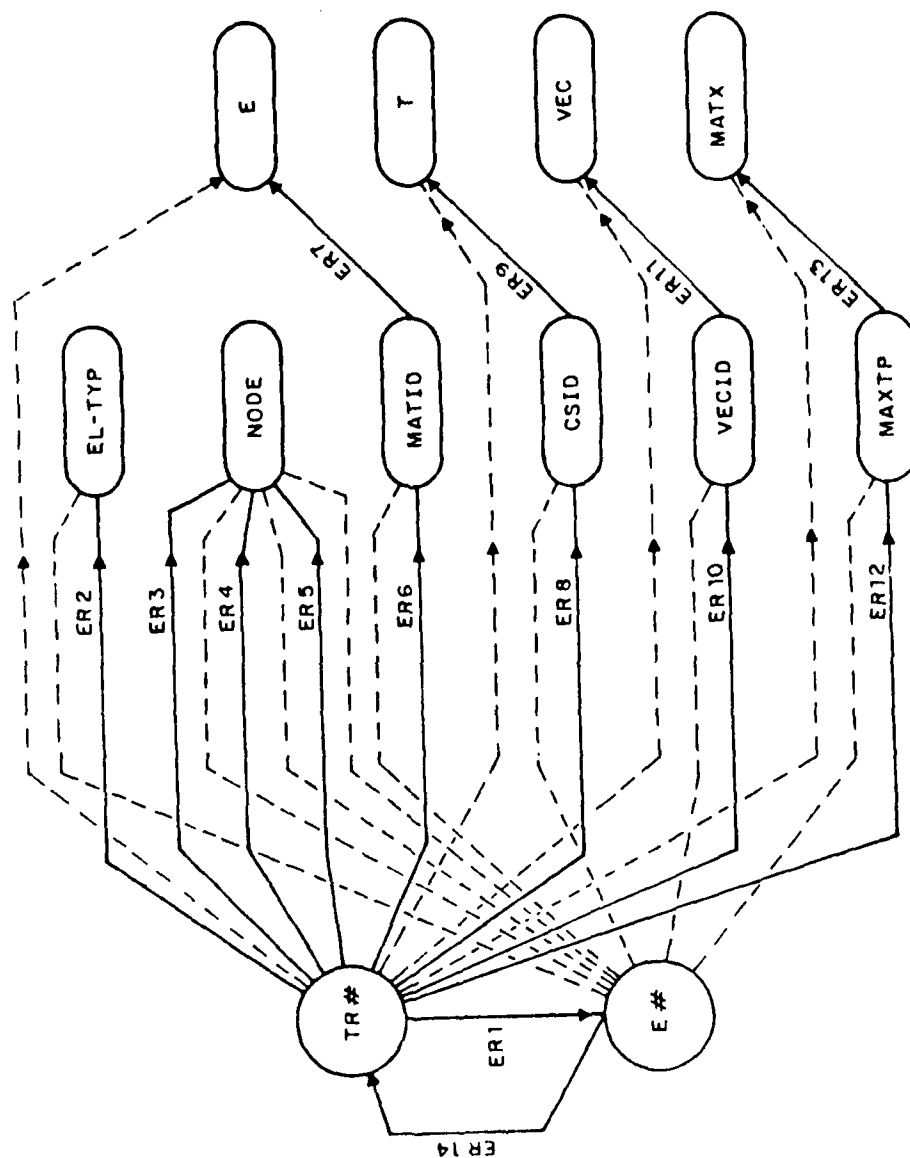


Figure 4.3.1 Diagram Representation of Elementary Relations



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	E #	EL TYP	TR #	N O D E 1 #	N O D E 2 #	N O D E 3 #	M A T I D	C S I D	V E C I D	M A T X I D	E	T	V E C	M A T X
1 E#		1*	1	1*	1*	1*	1*	1*	1*	1*				
2 EL-TYP														
3 TR#	1	1	0	1	1	1	1	1	1	1	1*	1*	1*	1*
4 NODE1#														
5 NODE2#														
6 NODE3#														
7 MATID											1			
8 CSID												1		
9 VECID													1	
10 MATXID														1
11 E														
12 T														
13 VEC														
14 MATX														

Figure 4.3.2 Connectivity Matrix C for Elementary Data



Table 4.3.1 Transitive Closure for Elementary Relations

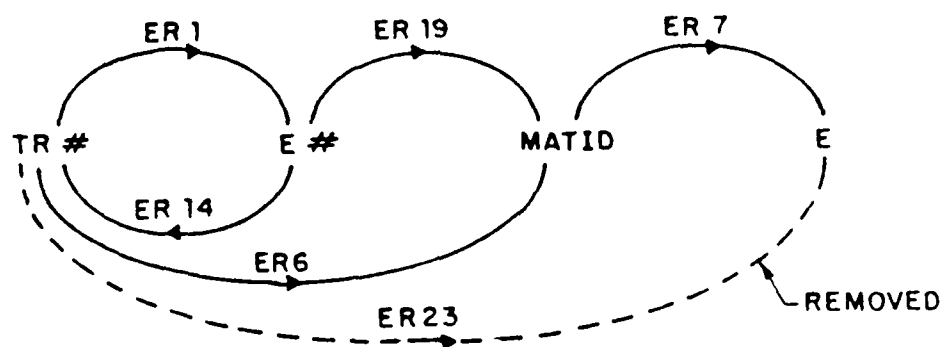
Derived Relations	Dependencies	Semantically Composition Meaningful	
ER15	E# → EL-TYP	E# → TR# → EL-TYP	YES
ER16	E# → NODE1#	E# → TR# → NODE1#	YES
ER17	E# → NODE2#	E# → TR# → NODE2#	YES
ER18	E# → NODE3#	E# → TR# → NODE3#	YES
ER19	E# → MATID	E# → TR# → MATID	YES
ER20	E# → CSID	E# → TR# → CSID	YES
ER21	E# → LOAD-TYP	E# → TR# → LOAD-TYP	YES
ER22	E# → MAXID	E# → TR# → MAXID	YES
ER23	TR# → E	TR# → MAXID → E	NO
ER24	TR# → T	TR# → CS-TYP → T	NO
ER25	TR# → VEC	TR# → VECID → VEC	NO
ER26	TR# → MATX	TR# → MATXID → MATX	NO



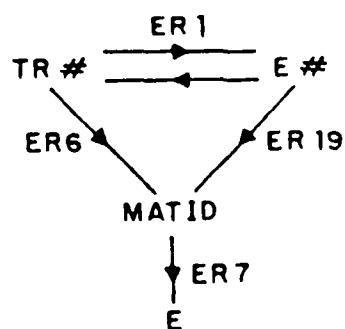
One method of removing redundant elementary relations is by finding a minimal cover (Vetter and Maddison, 1981). A minimal cover is a smallest set of elementary relations from which transitive closure can be derived. This method removes redundant relations by eliminating elementary relations which are composition of other elementary relations. The composition having maximum distance (defined in Section 3.3) is removed first and process is repeated iteratively to obtain a minimal cover. This method is illustrated in Fig. 4.3.3 and Table 4.3.2. However, the method is too tedious to use in practical situations where hundreds of elementary relations are formed.

We suggest an alternate methodology to select the elementary relations to form the conceptual data model. Information about processing sequence of data in finite element analysis and design optimization data should be used for eliminating redundant elementary relations. For example, consider the set of transitive closure of Fig. 4.3.3. E# in the example uniquely identifies all elements used in a structure, whereas TR# identifies only the element numbers in a triangular membrane element group. For example, to compute element stiffness matrix data, two processing sequences are possible. One processing sequence computes stiffness for all types of elements using E#. In this case, selection of relations ER1, ER19, and ER7 is more appropriate because material data is obtained in minimum number of accesses. On the other hand, if stiffness computation is carried out separately for each type of element (for example TR#), then alternate choice of elementary relations ER1, ER6, ER7 is suitable. Thus, in

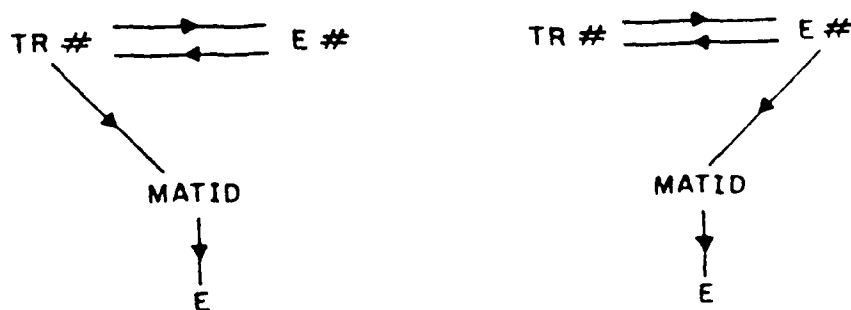




(i) Transitive Closure



(ii) Rearrangement



(iii) Minimal Cover

Figure 4.3.3 Digraph Representation of Minimal Cover



Table 4.3.2 Deriving Minimal Cover

Iteration	Composition ER <sup>z</sup> ER	$E_r$ $C(ER_i, ER_j)$	Removed $d_r$	Digraph $ER_n$	Minimal (Before Removed)	Cover
0	ER <sup>5</sup>	ER <sub>1</sub> ER <sub>2</sub> ER <sub>3</sub> ER <sub>4</sub> ER <sub>5</sub>	-- -- C(ER <sub>2</sub> , ER <sub>4</sub> ) C(ER <sub>1</sub> , ER <sub>3</sub> ) --	1 1 2 2 3 is not composed of other relations		NO
1	ER <sup>4</sup> ER <sub>1</sub>	ER <sub>1</sub> ER <sub>2</sub> ER <sub>4</sub> ER <sub>5</sub>	-- -- -- --	1 1 2 3		YES
1	ER <sup>4</sup> ER <sub>2</sub>	ER <sub>1</sub> ER <sub>2</sub> ER <sub>3</sub> ER <sub>5</sub>	-- -- -- --	1 1 2 2		YES



selecting a set of elementary relations to form a conceptual model of data, a certain amount of judgement is necessary on the part of database designer. By considering the processing sequence of data while removing redundant elementary relations will considerably reduce effort required to form a final set of elementary relations representing a conceptual data model for finite element analysis and design optimization data.

#### 4.4 Methodology to Design an Internal Model

Once we have a conceptual (theoretical) data model of finite element analysis and design optimization data, the next step is to design an internal model. Internal model shows how the data should be actually stored in a database using a database management software. An internal model specifies which attributes have to be combined together as a unit forming relations that are actually stored in a database.

In this section, a methodology to design an internal model is proposed. The methodology is based on two important aspects: (i) normalization of data, and (ii) processing requirement of data. Three normal forms of relations, i.e., 1st, 2nd and 3rd normal forms are used to arrange attributes selected from various domains. It is known that the use of normal forms of relations helps in avoiding inconsistency in data storage and update operations (Date, 1977). The second aspect of the methodology suggests the use of processing requirement of finite element analysis and design optimization data. The processing needs of analysis and design specify how various attributes should be derived from their underlying domains and combined together forming a n-ary



relation. Therefore, it is necessary to identify all the processes used in finite element analysis and design optimization computations.

After an internal model is developed, it should be verified for consistency with the conceptual model. Also, efficiency aspect of the model should be considered in the design. This aspect requires modification of the internal model to reduce the number of accesses to get the data. Finally, the internal model developed should be accommodated with a particular database management system that will actually be used to store and retrieve data. All these aspects of the methodology to design an internal data are discussed in the following paragraphs with the help of an example.

Design of an internal model to support element stiffness matrix generation process is described here. Methodology for designing internal models for other structural analysis and design process would follow similar steps. We assume that a conceptual model for element stiffness generation is already available. Our aim is to produce an internal model that is consistent with the conceptual model given by the following elementary relations:

ER1 (TR#, EL-TYP)	ER9 (NODE#, X)
ER2 (TR#, NODE1#)	ER10 (NODE#, Y)
ER3 (TR#, NODE2#)	ER11 (NODE#, Z)
ER4 (TR#, NODE3#)	ER12 (TR#, MATXID)
ER5 (TR#, MATID)	ER13 (MATXID, MATX)
ER6 (MATID, E)	ER14 (E#, NODE#)
ER7 (TR#, CSID)	ER15 (TR#, E#)
ER8 (CSID, T)	



Data needed for generation of element stiffness matrix are derived from various domains and represented in a single relation TRM-D as shown in Fig. 4.4.1. Our main intention is to get all the data required for generation of stiffness matrix for a triangular membrane element in one access or minimum number of accesses.

Observe from the relation of Fig. 4.4.1 that it is not in the first normal form. Therefore, this unnormalized relation should be replaced by a semantically equivalent relation in 1NF as shown in Fig. 4.4.2.

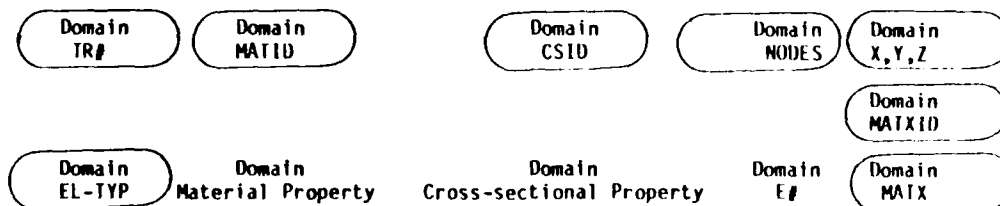
The following points have to be noted for normalization procedure in structural design context:

1. Row and column intersection of a relation in 1NF is atomic (i.e., consists of single values). Exception to these rules are vectors and matrices and they are considered to be atomic. In Figure 4.4.2, a set of values of MATX is identified as a unit.
2. Values for attributes are also derived from a non-simple domain. A non-simple domain is one which contains elements that themselves are relations. In Figure 4.4.2, STEEL refers to another relation which contain material properties.

The advantage of 1NF over the unnormalized relation are that operations required for application programs are less complicated and easy to understand.

The elementary relations identified earlier contain all the information required for generation of element stiffness matrices. This information should be reflected in some way in the internal model that is to be developed. There by we can ensure that internal model is





TR#	E#	EL-TYP	MATID	E	CSID	T	NODE#	X	Y	Z	MATXID	MATX
1	15	TRM3	STEEL*5	$10^8$	THICK*6	0.1	5 7 8	1. 3. 2.	5. 4. 6.	7. 8. 3.	STF*1	[...]
2	16	TRM3	ALUM*4	$0.9 \times 10^7$	THICK*4	0.2	12 14 18	5. 7. 3.	9. 4. 5.	8. 1. 8.	STF*2	[...]

Figure 4.4.1 A Tentative Internal Model

TR#	E#	EL-TYP	MATID	E	CSID	T	NODE#	X	Y	Z	MATXID	MATX
1	15	TRM3	STEEL*5	$10^8$	THICK*8	0.1	5	1.	5.	7.	STF*1	[...]
1	15	TRM3	STEEL*5	$10^8$	THICK*8	0.1	7	3.	4.	8.	STF*1	[...]
1	15	TRM3	STEEL*5	$10^8$	THICK*8	0.1	8	2.	6.	3.	STF*1	[...]
2	16	TRM3	ALUM*4	$0.9 \times 10^7$	THICK*4	0.2	12	5.	9.	8.	STF*2	[...]
2	16	TRM3	ALUM*4	$0.9 \times 10^7$	THICK*4	0.2	14	7.	4.	1.	STF*2	[...]
2	16	TRM3	ALUM*4	$0.9 \times 10^7$	THICK*4	0.2	18	3.	5.	8.	STF*2	[...]

Figure 4.4.2 Relation TRM-D in INF



consistent with the conceptual model. Internal model of Fig. 4.4.2 has all the attributes providing information for generation of element stiffness matrix. To check the consistency of this model, first we identify the key attributes. Candidate keys are compound, consisting of (E#, NODE#) and (TR#, NODE#). Primary key is selected as (TR#, NODE#). Secondary keys are TR#, E#, MATID, CSID, NODE#, and MATXID. These key attributes of the relation are consistent with those in the elementary relation. Secondly, we need to identify whether all the attributes in internal model and dependencies between them are consistent with the conceptual model. Observe that attributes NODE1#, NODE2# and NODE3# do not appear in the relation. Therefore, these three attributes should be included in the relation. Relation TRM-D is now written as

TRM-D (TR#, E#, EL-TYP, E, CSID, T, NODES#, NODE1#, NODE2#,  
 NODE3#, X, Y, X, MATXID, MATX)

The functional dependencies reflected by elementary relations ER1 to ER15 are satisfied in the internal model with the values shown in the Fig. 4.4.2. Therefore, at this instant the internal model is consistent with the conceptual model. However, it would be no longer consistent, if arbitrary changes in the values of the table are made. Also, note that many values in the relation TRM-D are redundant. These inconsistencies and redundancies occur because of the following anomalies in the 1NF:



1. INSERT operations: User cannot store details concerning a finite element without knowing at least one node of the element. The reason is that one part of the primary key (E#,NODE#), i.e, NODE# is not known.
2. DELETE operations: If user deletes a particular material type - MATID, the database automatically loses all the data about those finite elements using the material. Similarly, if user deletes a particular finite element, then database loses data about a particular material.
3. MODIFY operations: To change information about a particular element number, all rows containing that element number have to be modified. Otherwise functional dependency MATID  $\rightarrow$  E will not be valid any more.

Therefore it is not desirable to use the relation in Fig. 4.4.2 to represent the internal model. Modification of this relation to 2NF is necessary to avoid these anomalies in the storage operations. For this purpose, first we need to identify non-key attributes of TRM-D and check their dependence on the candidate keys. The non-key attributes are EL-TYP, E, T, X, Y, Z, MATX (refer to definition of full functional dependency). For example, consider the non-key attribute EL-TYP:

E#, NODE#  $\rightarrow$  EL-TYP

E#  $\rightarrow$  EL-TYP

NODE#  $\nrightarrow$  EL-TYP

Therefore,      E#, NODE#  $\nrightarrow$  EL-TYP



Thus, EL-TYP is not fully functionally dependent on (E#,NODE#). Note that E is transitively dependent on E# through MATID. Similar argument leads to

E#, NODE#  $\neq$  E, T, X, Y, Z or MATX.

Therefore, relation TRM-D should be converted into a set of semantically equivalent relations as follows:

TRM-D1 (TR#, E#, EL-TYPE, NODE1#, NODE2#, NODE3#,  
MATID, E, CSID, T, MATXID, MATX)

TRM-D2 (NODE#, X, Y, Z)

TRM-D3 (E#, NODE#)

The above three relations TRM-D1, TRM-D2 and TRM-D3 are all in 2NF because first two relations do not possess any compound candidate key and third relation has all keys. Note that by splitting the relation, TRM-D no information is lost and they still are consistent with the conceptual model. However TRM-D1 relation is still not satisfactory as it can lead to anomalies in storage operations as follows:

1. INSERT operation: It is not possible to store the fact that a particular material - MATID has a property - E without knowing at least one finite element using the material.
2. DELETE operation: If only one element is using a particular material and if that element is deleted, we loose all information about the material.
3. MODIFY operation: If several finite element use a particular material and if the property of the material is changed, then modification must be done to all the rows of the material used by those elements.



Therefore it is not feasible to use TRM-D1 relation in the internal model. Modification of this relation is necessary to 3NF to avoid anomalies in storage operation. Non-key attributes must be non-transitively dependent on candidate keys to avoid these anomalies. Observe from the relation TRM-D1 of Fig. 4.4.3 that attributes E, T, and MATX are transitively dependent on TR# through MATID, CSID, and MATXID, respectively. Removing these transitive dependencies, we get the following relation:

TRM-D4 (TR#, E#, EL-TYP, NODE1#, NODE2#, NODE3#, MATID,  
CSID, MATXID)

TRM-D5 (MATID, E)

TRM-D6 (CSID, T)

TRM-D7 (MATXID, MATX)

The above four relations together with TRM-D2 and TRM-D3 constitute the internal model for element stiffness matrix generation purpose. This internal model is consistent with the conceptual model identified earlier. Also, note that number of relations in the internal model is only 6 as compared to 15 elementary relations in the conceptual model.

In summary, the following steps are necessary to derive an internal model that is consistent with the conceptual model. Normalization procedures have to be adopted at each step to reduce redundancy and to eliminate undesired anomalies in storage operation. At each step unsatisfactory relations are replaced by others.

**Step 1.** Form relations with attributes derived from a set of domains.



TRM-D1

TR#	E#	EL-TYP	NODE1#	NODE2#	NODE3#	MATID	E	CSID	T	MATXID	MATX
1	15	TRM3	5	7	8	STEEL*5	$10^8$	THICK*8	0.1	SIF*1	[...]
2	16	TRM3	12	14	18	ALUM*4	$0.9 \times 10^7$	THICK*4	0.2	SIF*2	[...]

TRM-D2

NODE#	X	Y	Z
5	1.	5.	7.
7	3.	4.	8.
8	2.	6.	3.
12	5.	9.	8.
14	7.	4.	1.
18	3.	5.	8.

TRM-D3

E#	NODE#
15	5
15	7
15	8
16	12
16	14
16	18

Figure 4.4.3 Relations in 2NF



**Step 2.** Eliminate multiple values at row-column intersection of relation table. Vectors and matrices are considered to be single data items for this step.

**Step 3.** Result of Step 2 is the relations in the 1NF. Take projections of 1NF relations to eliminate any nonfull functional dependencies and get relations in the 2NF.

**Step 4.** Take projection of relations obtained in Step 3 to eliminate transitive dependencies to form relations in the 3NF. Thus a set of relations in the 3NF is the internal model.

The next question in the design of internal model is the efficiency. Is the internal model obtained by considering normalization process an efficient one to use? This question posed in another way -- is it possible to get the data required for a particular process in a minimum number of accesses to the database? If not, how to reduce the number of accesses to get the data. The answer to these questions is obtained by a study of the individual processes and their data requirement needs.

Again, consider the previous example of internal model consisting of relations TRM-D2, TRM-D3, TRM-D4, TRM-D5, TRM-D6 and TRM-D7. To access the thickness data  $T$  of a particular element  $TR\#$ , one has to first access the CSID data from TRM-D4, then retrieve TRM-D6 relation to get the required data. Thus two database accesses are required. Similarly, to get the coordinates of an element  $TR\#$ , first the node numbers of the elements have to be identified, then from relation TRM-D2



coordinates of nodes are retrieved. In this case, four database accesses are required including three accesses to relation TRM-D2. One possibility of reducing the number of accesses to the database is by combining relations TRM-D4, TRM-D2 and TRM-D5. Such combination increases efficiency at the expense of redundancy in data storage and anomalies in storage operations.

Therefore, it is suggested to use normal forms of relations for the internal model and introduce redundancy if required to improve efficiency at the actual implementation stage. But such redundant and unnormalized relations should be carefully noted to avoid erroneous operations on database.

#### 4.5 Some Aspects to Accommodate an External Model

One of the important requirements of a database is to provide facility for data retrieval by different application programs depending on their needs. Different application programmers can have different views of a database. Data structure as seen by an application program or interactive user is called an external data model. Generally, user's perspective of a database is only a subset of the actual contents of the database. Data retrieved from actual physical storage in the database undergoes transformation till it reaches the user. Transformation involves rearrangement of data from internal level to external level into a form acceptable to the application program. In the following paragraphs, considerations required while designing an external model suitable to structural design applications are given.



Some constraints have to be observed while designing an external model. Constraints arise while rearranging data from internal data structure to an external data structure. An important constraint is that internal data structure must be consistent with the conceptual data structure. Any retrieval and storage operations specified on external model must be correctly transformed into corresponding operations on the internal model and at the same time data must be consistent with the conceptual data model. Also design of the external model must fit the database management system capability. An example of how an external model is derived from an internal model is given below.

Suppose a particular user would like to know the coordinates of nodes of each triangular finite element for generation of element stiffness matrices. This means that the external model:

EL-CORD (TR#, E#, EL-TYPE, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)  
has to be provided for that particular user. Note that the external view EL-CORD contains data items from two different relations -- TRM-D4, TRM-D2 (refer Section 4.4). Therefore, a procedure is required to transform internal data model (relations TRM-D4, TRM-D2) to the external data model (relation EL-CORD). Data from TRM-D4 and TRM-D2 have to be rearranged to obtain relation EL-CORD. Procedure for rearrangement is formulated by using JOIN and PROJECT operations as follows:

TRM-A (TR#, E#, EL-TYP, NODE1#)  $\leftarrow$  TRM-D4

TRM-B (TR#, E#, EL-TYP, NODE2#)  $\leftarrow$  TRM-D4

TRM-C (TR#, E#, EL-TYP, NODE3#)  $\leftarrow$  TRM-D4

TRM-D (TR#, E#, EL-TYP, X1, Y1, Z1) = TRM-A\*TRM-D2



TRM-E (TR#, E#, EL-TYP, X2, Y2, Z2) = TRM-B\*TRM-D2

TRM-F (TR#, E#, EL-TYP, X3, Y3, Z3) = TRM-C\*TRM-D2

EL-CORD (TR#, E#, EL-TYP, X1, Y1, Z1, X2, Y2, Z2,  
X3, Y3, Z3) = TRM-D\*TRM-E\*TRM-F

NOTE: + indicates PROJECT; \* indicates JOIN

The above procedure (algorithm) yields EL-CORD relation. Observe from the algorithm that we did not modify the original relations TRM-D4 and TRM-D2 to retrieve the data required for a particular inquiry. The relations TRM-D4 and TRM-D2 are still consistent with the conceptual model. Therefore, pure retrieval operations for rearrangement of data does not cause any inconsistency in data values.

Now, consider the reverse process of transforming external data structure to internal data structure. Suppose, a particular user wants to insert the nodal coordinates of a finite element using the external view EL-CORD. Here, relation EL-CORD has the only key TR# and has no reference to the node number to which the element is connected. Insertion is not consistent with the conceptual model which requires that coordinate of nodes which are dependent on keys NODE#. This restriction is also reflected in the internal model -- TRM-D2 which requires NODE# as key values for insertion. Therefore, the transformation of relation EL-CORD into internal model is not possible. From this example, it follows that there are restrictions for rearranging data from external model to internal model.



#### **4.6 Methodology to Incorporate Large Matrix Data into a Database**

In finite element analysis and structural design optimization, we encounter problem of storage of large order matrices. These matrices are generally banded and sparse, and require careful consideration in organizing them in databases. This special nature of matrix data is unique to structural design database and therefore no attempts have been made to study this aspect in business database management area. However, a few matrix schemes have been implemented on disk files, but they are highly tailored to meet only specific application program needs and not suitable for general use. Consequently, there is a need for the development of a generalized and a new user-friendly technique to deal with large order matrices. In this section, we discuss various types of large order matrices and develop a suitable methodology for organizing them in a database.

##### **4.6.1 Identification of Matrices**

Various types of matrices are identified and defined below. Note that matrices considered for our purpose are of large order which implies a matrix  $A(m,n)$  where  $m$  and  $n$  about 1000 or more. For the purpose of our discussion, matrices are grouped into five types and are referred by the type number.

(i) Square Matrix  $A$

$$A \equiv a(i,j) \quad i = 1, 2, \dots, n, \quad j = 1, \dots, m$$

A square matrix  $A$  is symmetric if  $A = A^T$



A square matrix  $A$  is diagonal if

$$a(i,j) \neq 0 \quad \text{for } i = j$$

$$a(i,j) = 0 \quad \text{for } i \neq j$$

A square matrix  $A$  is upper triangular if

$$a(i,j) \neq 0 \quad \text{for } i \leq j$$

$$a(i,j) = 0 \quad \text{for } i > j$$

A square matrix  $A$  is lower triangular if

$$a(i,j) \neq 0 \quad \text{for } i > j$$

$$a(i,j) = 0 \quad \text{for } i < j$$

(ii) Banded Matrix  $A$

$$a(i,j) = 0 \quad \text{for } |i-j| > m$$

where  $m \ll n$ ;  $n$  = matrix size

$$a(i,j) \neq 0 \quad \text{for } |i-j| \leq m$$

Consider  $b_i = 1 + (j-1)$  for all  $i$ , where  $j$  is the column number for last nonzero entry in row  $i$ , then semi-band width  $B = \max b_i$  (refer to Fig. 4.6.1).

(iii) Hypermatrix  $H$

$$H \equiv h(k,\ell) \quad k = 1, \dots, p, \quad \ell = 1, \dots, p$$

where  $h(k,L) \equiv$  square nonnull matrix  $A$  for some,  $k, \ell$

$\equiv$  square null matrix  $A$  for some  $k, \ell$

and  $A \equiv a(i,j) \quad i = 1, \dots, m, \quad j = 1, \dots, m$

$A$  is known as submatrix of  $H$ .  $k$  and  $\ell$  are known as hyper-rows and hyper-columns (refer to Fig. 4.6.2).

$H$  is upper triangular if



$$h(k, \ell) \equiv A \text{ for } k < \ell$$

$$\equiv 0 \text{ for } k > \ell$$

H is lower triangular if

$$h(k, \ell) \equiv A \text{ for } k > \ell$$

$$\equiv 0 \text{ for } k < \ell$$

(iv) Sky-line Matrix S

For symmetric upper triangular matrix

$$S \equiv a(i, k) \quad i = 1, \dots, n, k = 1, \dots, n$$

$m_j$  = row number of first nonzero element in column  $j$ ;  $m_j$ ,

$j = 1, \dots, m$  define the skyline.

$(j - m_j)$  = column height

$$a(i, k) = 0 \quad \text{for } k > (j - m_j); \text{ (refer to Fig. 4.6.3).}$$

(v) Sparse Matrix P

$$P \equiv a(i, j) \quad i = 1, \dots, m, j = 1, \dots, n$$

P is sparse if

$a(i, j) = 0$  for most values of  $i$  and  $j$ . As a rule of thumb, only about 5 to 20% of the matrix contains nonzero values at scattered locations in the sparse matrix.

#### 4.6.2 Methodology for Design of a Numerical Model

We identified various types of matrices commonly encountered in finite element analysis and design optimization procedures. It is necessary to establish a methodology for organizing these matrices in a



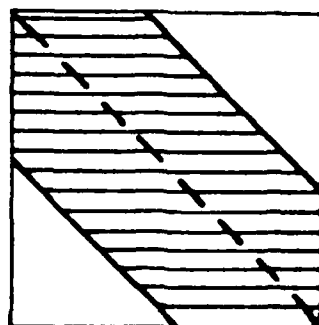


Figure 4.6.1 Banded Matrix

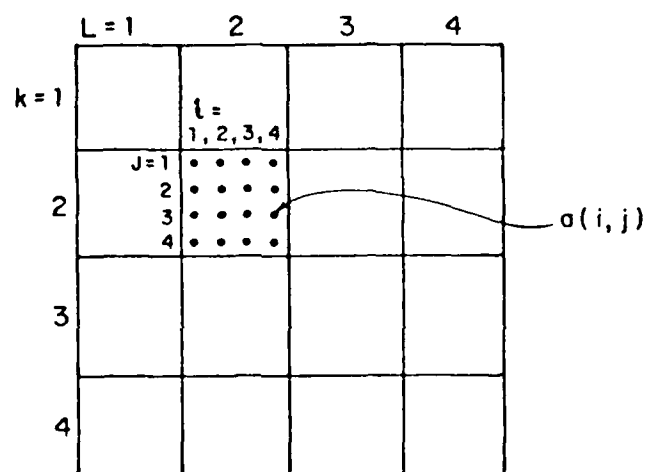


Figure 4.6.2 Hyper Matrix

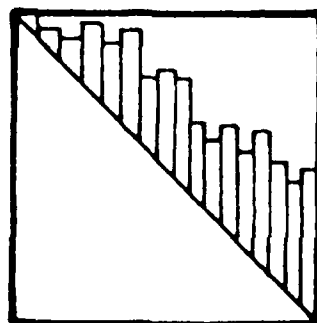


Figure 4.6.3 Skyline Matrix



database. A numerical model is proposed in this section which consists of conceptual, internal and external views of large matrices. Recall that a conceptual view represent inherent nature of data independent of any computer constraints. Therefore it is necessary to first study true nature of a large order matrix. Later, the internal representation of a matrix are considered to deal with storage efficiency, processing sequence, matrix operations, and flexibility of data modification. Since different applications and users view the same matrix in different form, suitable external views have to be provided.

Conceptually, a matrix is a two-dimensional array of numbers. These numbers appear in a certain pattern; e.g., square, sparse, symmetric diagonal, banded, lower triangular form, upper triangular form, unitary form, tridiagonal form, hyper matrix form and skyline form. A matrix is uniquely identified by a name. Rows and columns of the two-dimensional array are used for identification of data elements in the matrix. A conceptual view of a matrix is represented by the following elementary relations:

ER1 (NAME, MATRIX TYPE)

ER2 (NAME, NUM-OF-ROWS)

ER3 (NAME, NUM-OF-COLUMNS)

ER4 (NAME, ROW, COLUMN, DATA-ELEMENT-VALUE)

ER5 (NAME, NUM-OF-HYPER ROWS)

ER6 (NAME, NUM-OF-HYPER COLUMNS)

ER7 (NAME, HYP-ROW, HYP-COLUMN, ROW, COLUMN, DATA-ELEM-VALUE)

ER8 (NAME, BAND-WIDTH)



ER10 ( NAME, SUB-MAT-ROW-SIZE)

ER11 (NAME, SUB-MAT-COLUMN-SIZE)

ER12 (NAME, VECTOR OF SKYLINE-HEIGHT)

ER13 (NAME, HYP-ROW, HYP-COLUMN, NULL-OR-NOT)

The attributes of these elementary relations are self descriptive. These elementary relations completely define a matrix and provide the conceptual structure of the matrix. Note that the data values assigned to a matrix do not depend on whether a matrix is in banded, skyline or hyper matrix form. But they are located using on the row and column number of a matrix. Therefore, additional information such as banded, skyline, hyper matrix, bandwidth, and submatrix size are useful mainly to take advantage of the special nature of a matrix for storage and computational purpose.

Internal (storage) structure for large order matrices have to be developed which is consistent with the conceptual structure. The elementary relations defined above could be stored in a database, but it would require an awful number of accesses to get the required matrix data. Therefore, storage schemes have to be developed based on efficiency considerations. Also, storage space consideration is important to save disk space. Special nature of matrix, i.e., is sparse, dense, symmetric, should be used to provide storage efficiency. We can classify various matrix types considered in the previous section into two basic types - sparse and dense. Note that banded or diagonal matrices are not to be mistaken as sparse. Many possible storage schemes are available to store dense and sparse matrices. First, we consider storage of large order dense matrices.



Conventional storage schemes -- row-wise, column-wise, submatrix-wise are useful for storing dense matrices. Row and column storage are considered to be similar for purpose of our discussion. Thus, out of these storage schemes, only two schemes -- row-wise and submatrix wise are considered for evaluation. Figure 4.6.4 shows the row and submatrix storage schemes. Again, it is stressed here that we are considering the internal storage schemes and not the users view of a matrix - such as row-wise, column-wise, submatrix wise, skyline wise, or upper-triangular. Choice between these two storage schemes should be based on consideration of several aspects - storage space, processing sequence, matrix operation, page size, flexibility for data modification, ease of transformation to other storage schemes or user's views, number of addresses required to locate rows or submatrices, and availability of database management system support. These aspects are considered in detail below.

**Storage Space.** Row storage scheme can be used for square, banded and skyline matrix types. However, this scheme is not appropriate for hypermatrix. Symmetric, triangular, and diagonal properties of square matrix can be used in saving storage space if variable length of rows is used. Similar schemes can be used for banded and skyline matrices to store data elements that appear in a band or skyline column. Submatrix storage can be used for all matrix types. Submatrix storage is most appropriate for hypermatrix data. Both schemes have disadvantages when zero elements within a row or submatrix have to be stored.



1
2
3
4
⋮
n

1,1	1,2	...	1,n
2,1			
⋮			
m,1			m,n

Figure 4.6.4 Row and Submatrix Storage Schemes



**Processing Sequence.** Row storage requires assembly of matrices, storage and retrieval be made only row-wise. This becomes inefficient if row-wise processing cannot be made. Submatrix approach is suitable for all types of processing sequence -- row-wise, column-wise, or in any arbitrary order.

**Matrix Operations.** Operations such as transpose, addition, multiplications and solutions of simultaneous equations are frequently carried out at various stages of structural design. Row storage scheme is highly inefficient for matrix transpose when column-wise storage is required. During multiplication of two matrices A and B, a column of B can only be obtained only by retrieving all of rows of B. Therefore, row storage scheme become inappropriate for such operation. However, submatrix storage scheme does not impose any such constraints in matrix operation, thus provides a suitable internal storage scheme.

**Page Size.** A page is a unit or block of data stored or retrieved from memory to disk. A more detailed definition will be given in the next chapter. For a fixed page size, only a number of full rows or a number of full submatrices together with fractional parts of them can be stored or retrieved at a time. It is clear that fragmentation of rows or submatrices takes place depending upon the size of rows or submatrices. Large row size will overlap more than one page in memory and cause wastage of space. Submatrix scheme has the advantage of providing flexibility in choosing submatrix size to minimize fragmentation of pages.



**Flexibility for Data Modification.** For modifications of rows of a matrix both row and submatrix storage schemes are suitable. But row scheme would be more efficient than submatrix storage scheme. For modifications of a few columns of a matrix, row storage scheme requires a large number of I/O.

**Transformation to Other Schemes.** Submatrix storage scheme requires minimum number of data access to transform to column-wise storage scheme.

**Address Required.** Submatrix storage requires less number of addresses to locate data than row storage scheme provided submatrices are reasonably large.

Thus, for internal storage of large order matrices in a database, the above mentioned aspects should be carefully considered. It appears that both submatrix and row storage schemes can be appropriate for various applications.

In order that internal storage scheme be consistent with the conceptual model, we need to store additional information about the properties of the matrix. Those additional informations are given by the elementary relations ER1, ER2, ER3, ER5, ER6, ER9 to ER13. They can be combined together and stored in a relation with key attribute NAME. Relations required for internal storage are indicated in Fig. 4.6.5.

So far we considered schemes for internal organization of large matrices. Since different users view the same matrix in different forms - banded, skyline, hypermatrix, triangular, or diagonal - it is



NAME	MATRIX TYPE	NO. OF ROWS	NO. OF COLUMN	SUBMATRIX ROW SIZE	SUBMATRIX COLUMN SIZE	BANDWIDTH	VECTOR OF SKYLINE HEIGHT

HYP-ROW NO.	HYP-COLUMN NO.	NULL or NOT	SUBMATRIX
1	1		$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$
1	2		$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$
	...	...	...

Figure 4.6.5 Relations for Matrix Storage



necessary to provide the external views to suit individual needs. Unit of transactions on various views of a matrix may be row-wise, column-wise, submatrix-wise or data element wise. Internal scheme is submatrix-wise, where as external view need not be submatrix wise. Therefore, transformation are necessary to convert the internal matrix data into the form required for a particular user. Such a transformation is schematically indicated in the Fig. 4.6.6.

Next, we consider sparse matrix storage scheme. Several storage schemes have been suggested by Pooch (1973) and Daini (1982). They are bit-map scheme, address map scheme, row-column scheme, and threaded list scheme. Out of these row-column scheme is simple and easy to use. Also, row-column scheme can be easily incorporated into relational model. Therefore, this scheme can be considered for storing sparse matrices encountered in design sensitivity analysis.

Row-column storage scheme consists of identification of row and column numbers of nonzero elements of a sparse matrix and storing them in a table. This scheme provides flexibility in modification of data. Any nonzero value generated during a course of matrix operation can be stored or deleted by simply adding or deleting a row in the stored table. The row-column scheme is schematically shown in Fig. 4.6.7.

External view of row-column storage scheme can be provided through suitable transformation procedures. An external view of this scheme is shown in Fig. 4.6.8.



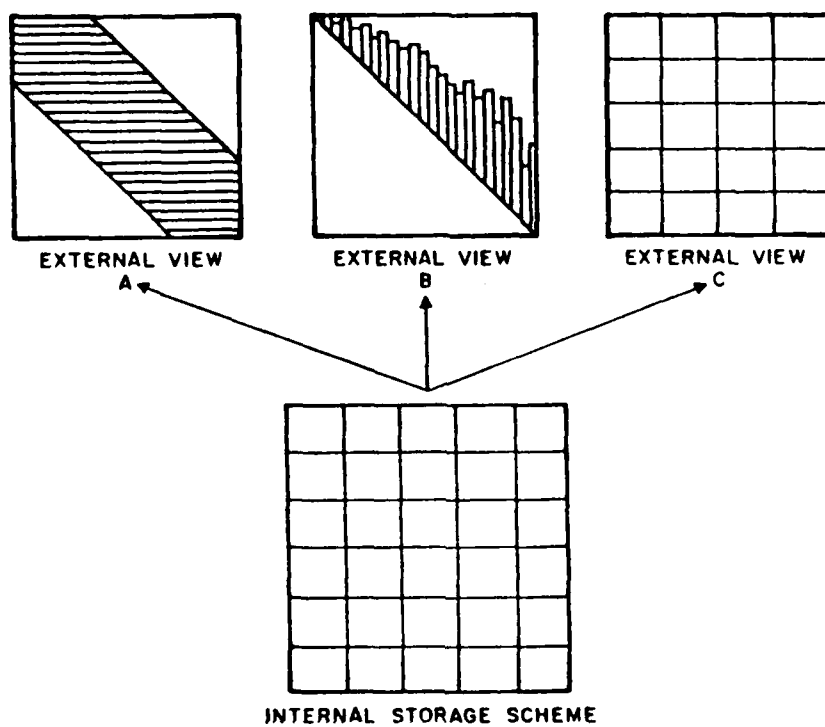


Figure 4.6.6 Transforming Internal Storage to External Views

ROW NO	COLUMN NO	VALUE

Figure 4.6.7 Row-Column Storage Scheme (Internal)



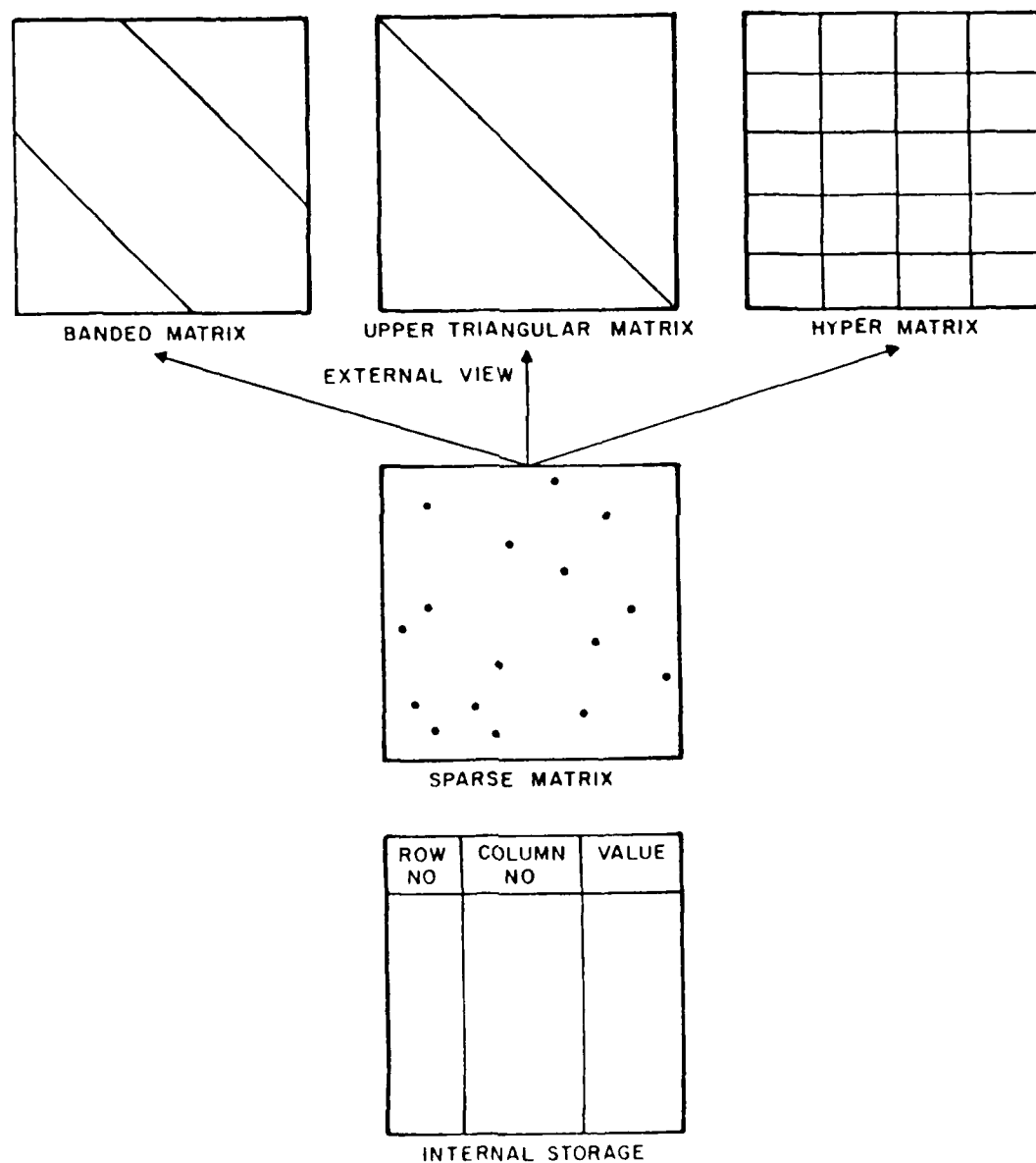


Figure 4.6.8 External View of Sparse Matrix



#### 4.7 An Algorithm or A Data Model?

So far we considered various methods and procedures for data organization, wherein, actual storage of data in a database was necessary. Many procedures in structural design, such as element stiffness matrix routines, generate huge amounts of data. Generally, it is unlikely that a user would want them to be stored in a database at the expense of disk space and data transportation time. The advantage of being able to query or possibly modify individual data items does not apply to stiffness matrices, which are more or less meaningless, except to the analysis program for which each matrix has been assembled. In order to save disk space and data transfer time, it is preferable to store only those data that are required for generation of element stiffness matrix. An algorithmic model is one where a data model is replaced by (i) an algorithm that generates the user requested information and (ii) a set of (condensed) data which will be used by the algorithm to generate the user requested information. Therefore, in an algorithmic model, data are not stored but are generated whenever they are needed. Algorithmic model suggested with a mixture of algorithms and stored data in a way that is most efficient for any given application.

Study of resource aspects must be made for deciding suitability of an algorithmic model or a data model. In cases where the items being modeled is rich in empirically derived data (for example, steel codes for allowable stress calculations) the algorithmic model uses a simple algorithm with a conventional data model. At the other extreme, where



all properties of an item can be generated (for example, element stiffness matrix calculations) the algorithmic model uses a complex algorithm with minimum data. A data model is preferable when storage capacity, processing costs, usage rates are high and time to transport data, rate of change of data are low. An algorithmic model is better if storage capacity, processing costs, usage rate of data are low and transport cost are high.

Methodology for constructing an algorithmic model is proposed based on (i) design of a suitable algorithm, and (ii) design of a (condensed) data model. Design of algorithms is dependent on the standard method of computational techniques used in finite element analysis and design optimization. These algorithms must be acceptable to all users of the model. Data transfer between algorithm and (condensed) data model can be provided through database management system support. Interface between algorithmic model and applications must be designed based on consideration of simplicity and ease of use. Design of (condensed) data model is dependent on the algorithm itself. If the condensed data model uses a conventional data model, then schemes for ensuring correctness of data values in the model must be provided. This is necessary because if this condensed data model is allowed to be modified arbitrarily then resulting data generated by algorithms will become meaningless. If several algorithms are in operation each using only a portion of condensed data model, then decomposition of the data model into several low level forms will enable efficient access of data values.



## CHAPTER 5

### DATABASE MANAGEMENT SYSTEM FOR STRUCTURAL DESIGN - A PROPOSAL

#### 5.1 Introductory Remarks

We need a software to use a database that has been designed based on the methodology given in the previous chapter. A software that handles requests of users to access and store data in a form compatible with data organized at various levels between physical storage level and application program level is called a database management system. A database management system conceals the complex data storage details and provide a simple view of database to the users. Such a software enables structural design application programmers and interactive users to store and retrieve required data in a simple way and thereby relieving them of unnecessary burdon of managing physical storage details.

This chapter is intended to identify various components of a DBMS and to formulate requirements of data definition, data manipulation and query language in view of implementation of a good DBMS for structural design optimization. A DBMS should have many components such as command processor, input-output processor, file operation routines, addressing and searching schemes, and memory management schemes in order to provide simple and efficient means of data storage and retrieval. Functions and requirements of these components are described in Section 5.2 with reference to structural design applications. In Sections 5.3 to 5.5



a proposal of syntactic rules (grammar) for languages used by structural designer to define data view, manipulate data and query data are given. Data definition, data manipulation and query language requirements are formulated. Finally in Section 5.6, existing database management systems are reviewed and their features tabulated. Requirements of a DBMS are also summarized in Section 5.7.

### 5.2 Components Required in a Database Management System

In this section components required in a database management system for structural design optimization are identified. Various components necessary in a DBMS and their functions are described. An overall view of the workings of a DBMS are given below to show main events that take place while an application program uses a DBMS:

1. An application program calls a DBMS to define a database, relation, and matrices.
2. DBMS checks the user given definition for syntax.
3. User requests DBMS to store and retrieve data.
4. DBMS transfers data from user buffer to disk and vice versa.
5. DBMS stores data on disk in files at addresses allocated for data.
6. A system buffer of DBMS is used as an intermediate storage to avoid too many disk I/O operation for data transfer between user buffer and disk.

If a DBMS is to only perform operations as described above, then the implementation for such a DBMS would be very simple. But the requirements of structural design application programmer and the usage



pattern are highly sophisticated and require a general purpose DBMS to satisfy their needs. Such a DBMS should have components -- command processors, input-output processors, addressing and searching routines, file definition and operation routines, memory management routines, relational operators, and security schemes. These are identified in Fig. 5.2.1. In the following subsections, functions and requirements of these components are described.

### 5.2.1 Command Processor

User given commands have to be checked before they are executed by a DBMS to avoid erroneous operations on a database. Commands of DDL and DML involve subroutine call statements, whereas query language commands contain character strings. It is necessary to verify these commands for syntax and also against illegal use of commands in any database operations. In the case of subroutine call statements, the number of arguments, type of arguments and value of arguments have to be verified. For example, if user has requested operations on a nonexistant database, command processor must issue an illegal operation flag. Interactive user is bound to commit a large number of mistakes while issuing commands spontaneously. Therefore, a more sophisticated command processor is required to verify query commands. Such a processor has to provide functions for verifying character strings, storing strings, separating data items from a command string, identification of integer, real and character data types in the string, finding the length of a command line, locating next item in a command



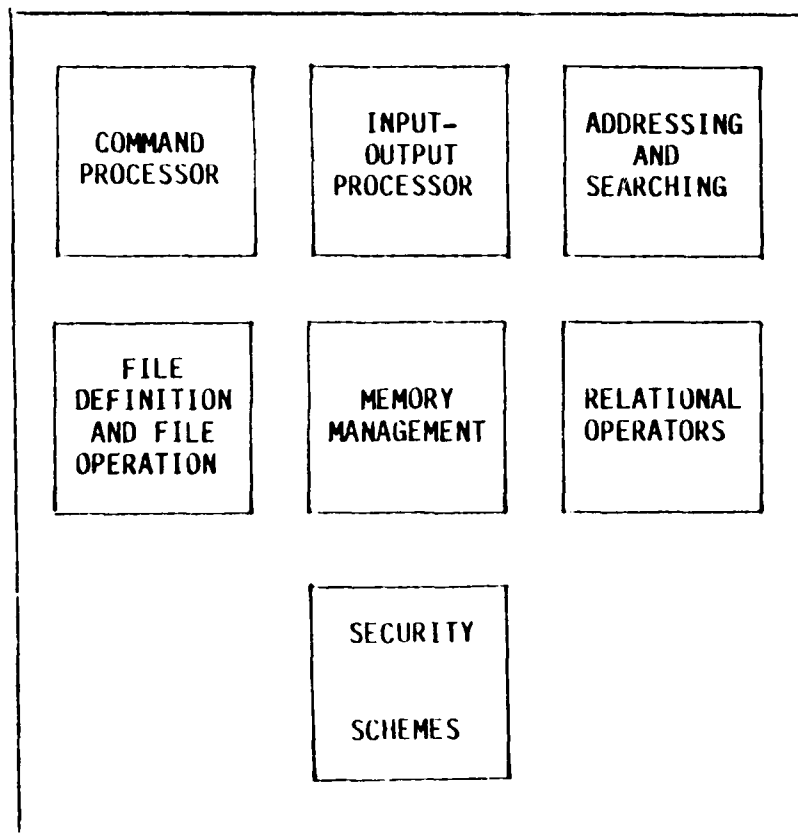


Figure 5.2.1 Components of a Database Management System



line, automatic generations of new commands, and finding repetitions of previous command line.

### 5.2.2 Input-Output Processor

Input-output processor is the basic component of a DBMS which handles all requests of the users to store, retrieve, modify and delete data. Storage and retrieval of data are done using data sets and relations as a unit. Data sets are nothing but sets of data stored in row, column or submatrix order. Relation is a two-dimensional table of data. User requests to store or retrieve portions (for instance a set of rows) of data set or relation are satisfied by providing data in the user buffer. Functions of I/O processor are to verify the correctness of data manipulation operations and to perform data transfer between user buffer and disk files. Existence of data sets, and relations are verified before any data is transferred between user buffer and database. If the data manipulation is based on primary keys, then I/O processor checks the existence of key values. Most of structural design applications, operates on rows of data set and relations. In such a case, I/O processor is required to verify the row numbers of data sets and relations. If data manipulation is based on certain conditions of data value in a data set or a relation (for example, modify coordinate values of nodes 5, 21, and 27) then I/O processor is required to provide capability for such manipulations.



### 5.2.3 Addressing and Searching

Input-output processors cannot directly get the required data from a data set or a relation till the address to the stored data is determined. Addressing routines determine the physical storage location for a data set or a relation given a particular row number or primary key values. There are several methods to determine the addresses. *Important methods that can be considered for implementation are indexing and hashing methods.* When index is used for addressing, address to blocks of records are maintained in a table. In hashing method, the primary key or row number is converted into a random number and is considered as the address for the data.

Index for addresses are generally large if database is big. Locating a particular index efficiently is important to save time of search. Out of several search techniques, B-tree search is popular. One index is placed at each node of a tree. Search begins at the top of a tree to locate a particular index. Depending on whether the given index is less than or greater than the index at a node, search is made toward left or right of the tree.

### 5.2.4 File Definition and File Operations

At the physical storage level, database consists of either a single stored file containing data or several files linked together as a unit. File definition and file operation routines are required in a DBMS. Function of file definition routine are naming of files, allocating logical unit numbers, specifying type of file access (random or sequential), specifying physical storage block size, etc.



Actual data and data definitions (name of data set, attributes, sizes) may be stored in a single file or can be separately stored in different files. File operations consists of opening, closing and deleting files. A file once opened for reading and writing should be closed at the end of file operations. File compression is needed to recover unused space.

#### 5.2.5 Memory Management

Structural design application programs use data from several data sets and relations at a particular design stage. Also, they use data of different portions of the same data set and relation in arbitrary sequence. In such situations, we are faced with the problem of accommodating the required data in the primary memory and at the same time reduce I/O activity to lessen computation time. Efficient use of primary memory is possible through judicious allocation of available space. Memory management scheme dynamically controls the available memory space. The memory is organized into pages (which is a unit of transfer between the database and primary storage) and page sizes are assigned. The size of the page is set to multiple of a physical record. Larger the page size, better the performance as less I/O is needed to get the required data. However, space may be wasted if there are too many partially filled pages. Small page size leads to increase in page replacement activity and maintenance of large page size table. Variable length pages require tedious programming effort.



Another important function of memory management routines is page replacement activity. Memory management scheme should keep count of the pages in the memory. Paging scheme may adopt some page replacement algorithm. Least recently used (LRU) page replacement is the most commonly used method. In LRU algorithm, a page counter is maintained for each page and updated each time the page is used. When a page is to be replaced, the page with the highest counter value becomes the candidate. A page replacement is needed when no free pages are available. A page not modified is over written instead of replacement.

Memory management scheme should be developed such that user has some control over the size of pages. This feature helps in determining the appropriate page size while operating on larger matrices of finite element analysis and design optimization problems. Fragmentation of large matrices on pages can be avoided by using page size in multiples of matrix size. Matrix operation algorithms for addition, multiplication and transpose by row operations can use page size in multiples of number of rows of a matrix. The algorithms that solve large order simultaneous equations by submatrix approach require at least a few submatrices to be present simultaneously in the memory. In such a case, allocation of one submatrix per page induces fewer page faults. This leads to reduction in I/O activity of iterative algorithms and brings down the execution time.

#### **5.2.6 Relational Operators**

A DBMS which operates on a relational database must have relational operators to manipulate the database. Relational operators manipulate



data in terms of entire sets or relations and not in terms of individual rows or columns at a time. Three types of relational operators are INTERSECT, PROJECT and JOIN. Each of these operators take either one or two relations as its operand and produces a new relation as its result. INTERSECT operator constructs a new relation by selecting rows of two relations satisfying certain conditions of specified attributes. The PROJECT operator forms a vertical subset of an existing relation by extracting specified columns and removing any redundant duplicate rows in the set of columns extracted. The JOIN operator, joins two relations, each having common column (attribute), and produces a new wider relation in which each row is formed by concatenating two rows, one from each of the original relation, such that two rows have the same value in those two columns.

#### **5.2.7 Security Schemes**

Structural design database is used by a number of application programs and users. It is necessary to ensure that database contents are not destroyed or manipulated by unauthorized users. Therefore, DBMS must have special scheme to ensure security of a database. Security of a database against unauthorized use can be provided by allocating password schemes to access contents of a database. Users of database are provided with read and modify passwords to use the database accordingly. These passwords can be assigned both at database level and individual dataset or relation level.



### 5.3 Requirements of Data Definition Language

Data definition language (DDL) is used to define database, relation and matrices. For the application programmer, DDL is a conventional language like FORTRAN; for interactive user it is the query language. It is necessary to build data definition language such that they do not contain reference to physical storage details on disk. Thereby any change in storage structure of data on disk will not force modification of application programs.

Data definition language for the application programmers consist of those declarative constructs of FORTRAN needed to declare database objects: Variables and arrays, FORTRAN data types, extensions to FORTRAN to support objects not handled by FORTRAN. Data is defined by application programmers using relations and/or matrices. Since FORTRAN does not provide facility to express relations and matrices, we need to provide extensions to FORTRAN to define data. Such an extension is possible through FORTRAN CALL statements and is provided as a part of DBMS. Arguments of the subroutine call statements, for example, specify database name, relation name, attribute name and size, etc.

Development of a DDL for structural design application should be based on several considerations. One of the major consideration should be to keep syntax of DDL concise and to be easily understood by application programmers. Furthermore, since, the DDL is used in structural design computing, all the data types of FORTRAN must be allowed -- integer, real, double precision and character. Data elements allowed by the DDL also include those of FORTRAN scalars and arrays.



DDL should support both relation and matrix definition. Relation definition require specification of relation names, attributes (names, type, and size), key attributes, and variable length attributes. Matrix definition requires specification of matrix name, matrix type, row size, column size and/or submatrix size. Large order matrices are organized in banded, hyper matrix or skyline form. Special facilities must be provided to define these large order matrices. There are many instances in structural design data where maximum size of data is known. For example size of stiffness matrix is known in advance. In such a case, provision in DDL for specification of maximum number of data occurrence will enable DBMS to conserve storage space and provide maximum efficiency. Another feature that should be included in DDL is data redefinition capability. This can be accomplished by providing several transformation of the same data to different application programs. It may be convenient, for instance, to represent a matrix in two different ways in different external views. In one view the matrix might be defined as two-dimensional array and in another view it might be defined as a vector. One important feature that should be incorporated in DDL of structural design application is compilation independent data definition facility. This means that DDL must have facility to define data types and relationships during run time. This feature is necessary because data definition in many instances are not known till certain stage of processing is complete. An example of this is number of degrees of freedom which is not known to define size of assembled stiffness matrix, till input data is processed. Finally, consideration



for development of DDL should include database security. Mechanisms for security such as read and modify password must be provided at both relation and matrix level.

Based on the above requirements a DDL for structural design application is proposed and its syntax given in the Appendix II. DDL syntax of Appendix II uses Backus-Naur Form (BNF) as a tool for defining syntax of languages. BNF is a common tool for formally describing a programming language syntax. A syntax is nothing but grammar of a language. The data definition language has following features:

1. Database definition with provisions to define global and local databases. Each database can be either permanent or temporary. Temporary meaning that database is deleted at the end of execution.
2. Database user identification provision. Database can be used either by a owner or a user. Database access is defined either by read or modify rights.
3. Data set definition facility with integer, real and double precision data types. Scalars, vectors and matrices can be defined.
4. Relation with attributes of interger, real and double precision data types. Attributes can be scalars, vectors, and matrices. Provision for key attributes and variable length attributes are made.
5. Termination of data set definition and data redefinition facility is available.



6. Numerical data definition with provision to define square, triangular, banded, hyper and skyline matrices is provided.

#### 5.4 Requirements of Data Manipulation Language

Data manipulation language (DML) is used to store, retrieve, modify and delete data in a database. For application programmers DML is provided through subroutine call statement; for interactive user it is provided through query language.

Development of a DML for structural design application should be based on several considerations. The DML commands should be simple as they are frequently used in an application program. The DML should not contain any reference to the storage structure details. DML should have capability to access data from different databases. Relations and matrices are frequently accessed and stored row-wise. Therefore, commands in DML must facilitate this operation. Also, it should be possible to store and retrieve rows in a sequential order or in a random order. Further, each row can contain data from a set of attributes each belonging to different data type -- integer, real, double precision. Therefore, commands should be designed to accommodate multiple data types for data manipulation operations. Many structural design application programs require data of a particular attribute of a relation. For example, some programs need only degree of freedom attributes in a node-coordinate-degree of freedom relation. Consideration in design of DML should include data manipulation in terms of single attributes or a set of attributes. Further, it should be



possible to select any required values from an attribute that satisfies certain conditions. Special commands must be provided in DML to specify conditions on data values that are to be manipulated.

Further consideration in the design of DML is based on matrix manipulation requirements. Large order matrices are generally stored and retrieved row-wise, column-wise and submatrix-wise. Data manipulation commands must include operators to manipulate matrix data in terms of rows, columns and submatrix order. Matrix operations such as transpose, multiplication require column-wise retrieval of data stored in row-wise pattern. Special provisions should be made in the commands to indicate null rows (zero elements in a row), null columns and null submatrices.

There are a number of other considerations for designing DML. Commands should include utility and data definition list facility. Utility commands of DML are open database, close database and error display commands. Commands should be able to provide facility for opening and closing of a number of databases at various stages of execution. Data definition list command will enable application programmer to check and use detailed definition of data objects stored in the database. Error display commands facilitate application programmer to investigate the cause of errors.

Based on the above requirements a DML for structural design application is proposed and its syntax given in Appendix III. DML syntax of Appendix III is given in Backus-Naur Form. Description of BNF are same as those given for DDL. The data manipulation language has the following features:



1. Commands to open and close a database.
2. Retrieval command. This has facility to retrieve data set and relation.
3. Store command to fill data set and relations with values.
4. Command to delete parts of data sets and relations.
5. Command to modify parts of a data set or a relation.
6. Remove command to eliminate a data set or a relation.
7. Command to copy data from one data set or relation to another.
8. Store, retrieve, modify and delete commands for matrix data.
9. Rule command to specify conditions on data values that have to be retrieved.

### 5.5 Requirements of Query Language

Structural designers often want to use a database interactively to find out various parameters of design and to modify them by using simple commands. Query language provides a simple set of commands to interactively define and manipulate data in a database. It can be used by any nonprogramming user and does not require knowledge of high level languages like FORTRAN. Data definition of query language includes database, relation and attribute definition, rule specification, and authorization procedures. Data manipulation of query language includes store, retrieve, modify and delete commands. In addition to these commands relational operators like INTERSECT, PROJECT and JOIN are required.



A query language should satisfy several requirements. These are data independence, simplicity, nonprocedurality, extendability and completeness. Data independence refers to independence of logical data structure and storage structure definition. Query language should not contain any reference to storage structure. Nonprocedurality means that user should be allowed to give commands in any arbitrary order and should not restrict them to follow procedures to query a database. Query language should be easily extendable to incorporate any special function into it. Query language should be complete in the sense that it possesses all the required commands to query all types of data in the database. Query commands must be general and not limited to any special case. Query of large order matrices requires special features in query language so that data can be displayed in parts.

General syntax of a query language is:

<command> <Expression ; clause> <conditional clause>

The command is a name interpreted by a DBMS to execute certain procedure for defining and manipulating data. Some typical commands required for structural design are SELECT, LIST, CHANGE, RENAME, OPEN, CLOSE, LOAD, DEFINE, and EXIT. The second component is expression clause which is a group of words specifying names of relation and attributes. The third component, conditional clause, allows a user to specify a condition on the data for which command is executed. The conditions may be, for example, GT.100; LT.20, ROWS.EQ.10.



### 5.6 A Review of Database Management Systems

In this section, a review of existing database management systems is made. This review (Sreekanta Murthy and Arora, 1985a) enables us to evaluate, select and modify a database management system so that it could be used in computer-aided structural analysis and design optimization. The following fifteen database management systems are reviewed. The capability of these systems are emphasized and important features are tabulated.

**DELIGHT** - Design Language with Interactive graphics and a Happier  
Tomorrow

**DATHAN** - A data handling program for finite element Analysis

**EDIPAS** - An Engineering Data Management System For CAD

**FILES** - Automated Engineering Data Management System

**GIFTS** - GIFTS Data Management System

**GLIDE** - GLIDE Language with Interactive graphics

**ICES** - Integrated Civil Engineering System

**IPIP** - Information Processor for IPAD

**PHIDAS** - A Database Management System for CAD

**REGENT** - A System for CAD

**RIM** - Relational Information Management System

**SDMS** - A Scientific Data Management System

**SPAR** - SPAR database management system

**TORNADO** - A DBMS for CAD/CAM system

**XIO** - A Fortran Direct Access Data Management System



**DELIGHT.** It stands for Design Language with Interactive Graphics and a Happier Tomorrow (Nye, 1981; Balling, Piester and Polak, 1983). In its philosophy, the DELIGHT system is very close to the GLIDE system (Eastman and Henrion, 1980). DELIGHT is an interactive programming language. It has good extension and debugging capability. It provides high-level graphic commands, a built-in editor and a well-defined interface routines. A single statement, procedure or part of an algorithm can be tested without having to write and load/link a program. The system relies on virtual memory management of the operating system. It is difficult to use the system with large scale programs. Multiple users are not allowed in the system.

**DATHAN.** It stands for data handling program. It is written mainly for finite element analysis applications (Sreekanta Murthy and Arora, 1983). The program has some basic in core buffer management scheme. It has capability to store permanent and temporary data sets. Substructure files can be arranged quite easily with same data set names for different substructures. Both integer and real data types can be handled. Drawback of the system is that the user has to keep track of the location from which a new data set has to begin. The system has FORTRAN data manipulation commands which are simple to use.

**EDIPAS.** It stands for Engineering Data Interactive Presentation and Analysis System, (Heerema and van Hedel, 1983). It is a tool for data management, analysis, and presentation. The data management part provides a utility to initialize a project database, input programs to



load data from files into database under user controls, and a set of routines to extract data from and load data into database in a controlled way. EDIPAS allows users to name a database, a data structure, and data entities. It allows user to employ one or more hierarchical levels. The data is stored in entities called blocks. A data block allows matrices, single values and characteristic values as data elements. A database administration support provides initialization of database, access to users, deletion of data structures, audit database contents, and back-up facility. The system does not have data redefinition facility. Improvements are being done to include redefinition facility in order that the data structures and their levels can be manipulated. Extension of authorization provision from database level to the level of data element is being incorporated.

**FILES.** It is an automated engineering data management system (Lopez, 1974). It is extremely flexible with respect to the definition of a database and methods of accessing it. Information storage and retrieval may be performed using problem-oriented languages. Hierarchical data structure is provided. For example matrix type of data encountered in finite element application can be organized using hierarchical data structure. The first two levels in hierarchy may contain pointers to the third level containing actual matrix data. The program allows dynamic memory allocation. Data transfer takes place between FORTRAN common block and database. FILES has a data definition language. The system does not have data mapping language to specify mapping of data items and arrays to an external device. The data



definition language (DDL) depends on the problem oriented language (POL). Therefore DDL cannot be used independently. The system requires a distinct data management compiler.

**GIFTS.** It is an interactive program for finite element analysis (Kamel, McCabe and Spector, 1979). It is a collection of modules in a program library. Individual modules run independently and communicate via the unified database. The database manager processes requests for opening a file, closing a file, storing data set in a file, and retrieving data set from a file. The program has memory management scheme. Each data set is stored in a separate random access file. Paging is carried out within the working storage. A unique set of four routines is associated with a data set for opening and initializing the working storage, for reading a data set, for creating/modifying the data set, and for realizing the working storage. Drawbacks of the system is that every new data set requires created four new routines to be written. Each data set is associated with a separate common block, thereby increasing the number of common blocks in the system. The data manager is application dependent and cannot be used as a stand alone system.

**GLIDE.** It is a context-free database management system (Eastman and Hennion, 1980). It is designed to provide a high level facility for developing individualized CAD system. It can be viewed as a language, a database management system, and a geometric modelling system. It allows users to define new record types known as FORM that consist of a set of



attribute field. It provides primitive data type set to organize a database. It provides excellent geometric modelling system or a graphic system. Drawback of GLIDE is that it does not allow multi-dimensional arrays.

**ICES.** Integrated Civil Engineering System is a computer system designed for solving civil engineering problems (Roos, 1966). ICES consists of a series of subsystems each corresponding to an engineering discipline. It provides a Problem Oriented Language which can be used to write subsystem programs (e.g., coordinate geometry program, stress analysis program). Command Definition Language is used by a programmer to specify the structure and required processing for each subsystem commands. A Data Definition Language is used to specify the subsystem data structure. It uses its own programming language called ICETRAN (ICES FORTRAN) and has a precompiler which translates ICETRAN to FORTRAN statements.

Dynamic data structuring capability is provided in the system which helps to organize dynamic arrays in the primary memory. Hierarchical data structure is used for data modelling. Three hierarchical levels: equivalence class, members, and attributes are provided. Data is stored on secondary storage using random access files. Data management program uses buffers to convert logical records to physical records. Identifier is supplied by the programmer which is a pointer giving the position on secondary storage of physical record. The programmer has a choice to store data using dynamic arrays or using data management system depending on amount and use of the data.



Drawback of the system is that it uses precompiler ICETRAN to convert to FORTRAN program instead of directly to machine language. Physical storage of data requires knowledge of address and pointers which the programmers have to give. Only three levels of hierarchy is adopted and it is difficult to extend to many levels of hierarchy.

**IPIP.** It is a state-of-the-art database management system satisfying engineering requirements (Johnson, Comfort and Shull, 1980). It offers a number of capabilities ranging from support for multiple schemas and data models to support for distributed processing, and data support for distributed processing, and data inventory management. An integrated software architecture supports all user interfaces: programming languages, interactive data manipulation and schema languages. IPIP supports a multiple-schema architecture of ANSI/SPARC database group. Three types of schemas -- conceptual, external and internal schemas are supported. IPIP schema and data manipulation languages exhibit a high degree of integration and compatibility. The logical schema supports both the network and relational data models, and, functionally, the hierarchical data model. The internal schema of IPIP is written using the internal schema language compiler. The internal schema language overlaps that of the logical schema language to the greatest practical extent to minimize the amount of schema language with which the administrator must deal. IPIP software subcomponents consists of user interface, and data manager. Software of user interface is made of precompilers, query processor and compilers. Data manager software is made of scheduler, message



procedure interface processor, common semantic processor, database control subsystem, data manipulation subsystem, record translator, presentation service, access module, resource manager and stubs.

**PHIDAS.** It is a data management system specially designed for handling a collection of structured data on minicomputers (Fischer, 1979). The architecture of PHIDAS is in accordance with the ANSI-3 schema. It has an external subschema based on network model of CODASYL and an internal schema for physical tuning particularly suited for engineering database. The data description language is provided to describe schema and sub-schema. PHIDAS also has a storage structure description language. Data manipulation language is FORTRAN call statements to subroutines. Drawback of the system is that it is difficult to represent matrix type data.

**RIM.** It stands for Relational Information Management system (Comfort, Erickson 1978). RIM has capability to create and modify data element definition and relationships without recompiling the schemes or reloading the database. RIM provides capability to define new types of data for use in special application such as graphics. RIM supports three types of data: real, integer, and text. Data definition and data manipulation languages are available to define or manipulate relations. The user has capability to project, intersect, join and subtract relations. RIM has good query language. RIM's modification commands permit the user to update relation definition, change data values, attribute names, delete tuples and delete the entire relation.



Utility commands such as LOAD, and EXIT are provided to load a new database and close an existing database. Drawback of RIM is that it does not allow relation having row size more than 1024 computer words. The application oriented FORTRAN call statements do not have capability to define attributes, relations, rules, etc., required in defining a schema. The system does not support management of a temporary database. Simultaneous operations on a number of databases is not possible.

**REGENT.** It is a system for the support of computer-aided design (Leinemann and Schlechtendahl, 1976). The main goal of the development was to provide a so-called "system nucleus" in the sense of ICES. Improvement claimed for the system is that it has a powerful base language PL/1 instead of FORTRAN. Interactive use has been considered in system development. The database management of REGENT provides facilities to compress a database, copy data between databases, and to change name and size of data elements. The database of REGENT is not a database in the usual sense. It is some sort of partitioned data set concept, built up using a tree structure of sequential files, but the internal structure of these files is known only to those programs that use them.

**SDMS.** It is a database management system developed specifically to support scientific programming applications (Massena, 1978). It consists of a data definition program to define the form of databases, and FORTRAN compatible subroutines to create and access data within



them. Database contains one or more data sets. A data set has form of a relation. Each column of a data set is defined to be either a key or data element. Key must be a scalar. Data elements may be vectors or matrices. The element in each row of the relation forms an element set. Temporary database capability that vanishes at the end of a job is provided. A scientific data definition language provides a program-independent data structure. Both random and sequential access of data set is possible. Data elements include scalars, fixed and variable length vectors, fixed and variable-size matrices. Data element types include text, real and integer. Drawback of the system is that it does not have a query language. Generalized database load/unload is not available. Double precision data type is not allowed. The system is implemented only on Cyber series computers.

**SPAR.** The computer program is a collection of processors that perform particular steps in finite element analysis procedure (Whetstone, 1977). The data generated by each processor is stored on a database compiler that resides on an auxillary storage device. Each processor has a working storage area that contains the input and the computed data from the processor. Allocation of spaces in the storage area is a problem dependent and is dynamically allocated during execution. Data transfer takes place directly between a specified location on disk using a set of data handling utilities. SPAR database complex is composed of 26 data libraries or data files. Libraries 1 to 20 are available for general use. Libraries 21 to 26 are reserved for temporary and internal use. The database manager uses a master directory



to locate the table of contents which in turn is used to locate the data sets in the database. Physically, the auxiliary storage is divided into sectors of fixed size and each read/write operation begins at the beginning of a sector. Drawback of the system is that it does not provide either hierarchical or relational data structure. Excessive fragmentation may take place if the sector size does not happen to be an integral multiple of the data that is stored.

**TORNADO.** It is a DBMS system developed for CAD/CAM application (Ulfsoy, Steiner and Oian, 1979). It is a CODASYL network system written in FORTRAN and is very useful for handling complex data structures. It handles variable object length and dynamic length records. System allows different data types - integer, real, character, double precision, double integer, complex and logical data. The system has easy to use data definition language and data manipulation language. TORNADO system is highly portable. Data in the database can be accessed by name. There is no restriction on data set types and allows many-to-many relationships. Drawback of the system is that the size of a data object defined by the system is limited by the largest integer value that can be represented in the computer. The size of the database is limited by the maximum size of a file. A multi-file version is not available. The database cannot be used by multiple users at the same time.

**XIO.** It is a set of subroutines that provides generalized data management capability for FORTRAN programs using a direct access file



(Ronald, 1978). The system allows arrays of integer, real double precision and character data storage. Both random access and sequential access of data is provided. Variable length record I/O is allowed in the system. Bit map scheme is used to identify the unused space for storage of data to minimize disk storage requirement. The program allows restart facility using saved file following completion of a partial execution or after a program termination. The system at present is only implemented on IBM360 or DEC PDP11 computing systems. The system does not provide data definition language. It does not provide either hierarchical or relational data structures.

### 5.7 Summary of Requirements of a DBMS

Capabilities of various systems are summarized in the Table 5.7.1. We identify the following requirements of a DBMS for computer-aided structural design optimization (Sreekanta Murthy, Shyy and Arora, 1985):

1. As FORTRAN is the host language for majority of the engineering applications, it is necessary to provide application interface with DBMS through standard FORTRAN statements.
2. Data model provided in DBMS must be easy to understand and apply for design application programmers and users. Data model must be flexible to suit the requirements of different applications. Relational and numerical data models are desirable.
3. Since the system will be used for design optimization in multidisciplinary environment, it must be an application-independent DBMS.



4. Design data consists of arrays and matrices to a large extent. Often matrix data are in the form of banded, submatrix, and triangular. A suitable data model is therefore required to organize matrix data.
5. DBMS should be able to deal with various data types such as characters, short integers, long integers, single precision, double precision, and complex numbers.
6. Speed of storage and retrieval is one of the most important requirements of a DBMS in design optimization. Short access time will considerably reduce the total execution time in an iterative nature of design process.
7. Simple to use data definition and data manipulation languages are required. Applications often define data dynamically. For example, size of various matrices, length of data, etc., are not known at compilation time. They are also modified frequently. It is necessary to provide data definition capability to cater to these special needs.
8. DBMS should provide additional capability to organize the available primary memory. A suitable memory management scheme should be incorporated.
9. A good query language is required for interactive design applications. Query language should be general to cover all applications.



10. Provision for managing a temporary database will considerably help designers in evaluating trial designs and transferring the acceptable final design to a permanent database. Temporary databases will also be needed in iterative design optimization process. Thus, DBMS must be able to handle multiple databases simultaneously.



Table 5.7.1 Features of Various Database Management Systems for Engineering Applications

NO	DBMS	HOST LANGUAGE	EXTERNAL MODEL	INTERNAL MODEL	DATA DYNAMIC DATA DEFINITION		DATA DEFINITION LANGUAGE	DATA MANIPULATION LANGUAGE	MEMORY MANAGEMENT	APPLICATION INTERFACE	INTERACTIVE CAPABILITY	MATRIX ORGANIZATION	GEOMETRIC DATA CAPABILITY		SIMULTANEOUS ACCESS TO MULTIPLE DATABASE FACILITY	MULTIPLE USER TEMPORARY DATABASE
					DATA DYNAMIC DATA DEFINITION	DATA DEFINITION LANGUAGE	DATA MANIPULATION LANGUAGE	DATA MANIPULATION LANGUAGE	MEMORY MANAGEMENT	APPLICATION INTERFACE	INTERACTIVE CAPABILITY	MATRIX ORGANIZATION	GEOMETRIC DATA CAPABILITY	GEOMETRIC DATA CAPABILITY		
1	DELIGHT					AS	AS	AS	A	A	A	A	A	A		
2	DATMAN	F		D	A	A	A	A	A	A		AS				A
3	EDIPAS		M			A	A	A		A	A	AS	A			
4	FILES	F	M			AS	A	A	AS	A		A			A	A
5	GIFTS	F		D			A	A	A	A		AS	AS	AS		A
6	GLIDE					AS	AS	AS		A	A			A		
7	ICES	O	M, D		AS	AS			AS	AS						
8	IPIP		M, N, R	A	A	A	A	A		A	A		A	A	A	A
9	PHIDAS	F	M	A		A	A	A	A	A	AS		A			
10	RIM	F		R		A	A	A	A	A	A	A	A	A		A
11	REGENT	PL			AS	AS			AS	AS						
12	SOMS	F		R		A	A	A		A		A		A	A	A
13	SPAR	F		D		A	A	A	AS	A		A		A		
14	TORNADO	F	M		AS	A	A	A		A				A		
15	X10	F					A	A	AS			A				

A = Available  
AS = Available but SpecializedM = Hierarchical  
F = FORTRANN = Network  
PL = PL/IR = Relational  
D = Others

O = Data set



## CHAPTER 6

### IMPLEMENTATION OF A DATABASE MANAGEMENT SYSTEM -- MIDAS

#### 6.1 Introductory Remarks

A database management system - MIDAS<sup>\*</sup> has been implemented for finite element analysis and structural design optimization applications (Sreekanta Murthy, Shyy, and Arora, 1986). The MIDAS implementation is based on the requirements of a database management system given in Chapter V. MIDAS has two subsystems - MIDAS/R and MIDAS/N. These subsystems are capable of organizing data of relational and numerical models, respectively. The system has been installed on PRIME computer system. It was decided to use an existing package as much as possible. RIM is the most advanced system available for scientific database management. It supports relational data model facility, so it was decided to see if the system could be extended to satisfy the requirements stated in Chapter 5. It was found difficult to extend RIM to have multiple databases, to organize large matrices, and to be efficient in handling large data sets and large memory. It essentially meant rewriting the memory management, and data definition and manipulation parts. So, it was decided to use RIM as is but add new data definition and data manipulation subroutines that could be called

---

<sup>\*</sup>(Management of Information for Design and Analysis of Systems)



from a FORTRAN application program. Also a memory management routine is added. This subsystem is called MIDAS/R which stands for MIDAS-Relational Data Management System.

A second subsystem called MIDAS/N was designed (Shyy, 1985) which stands for MIDAS-Numerical Data Management System. MIDAS/N supports numerical data model facility. This subsystem can handle multiple databases, small and large matrices, and small and large memory environment. Large matrices such as rectangular, square, upper-triangular, lower triangular and hypermatrices can be defined in the database. Matrix data can be arranged in row, column or submatrix order. MIDAS relieves the burden of managing data for application programmers by providing user-friendly application commands. The system has sophisticated interactive commands to query the database. The MIDAS system can be used either interactively or through application programs. The implementation details of MIDAS/R and description of MIDAS/N are given in Sections 6.2 and 6.3, respectively.

## 6.2 Implementation of MIDAS/R

In this section, capabilities, database organization, data definition, data manipulation and query commands of MIDAS/R are described. Also, details of the system architecture are given. MIDAS/R database management system is based on relational data model. The system is developed by modifying and extending the RIM program (Relational Information Management System).



### 6.2.1 Capabilities of MIDAS/R

MIDAS/R is written in FORTRAN-77. The system does not have any machine dependent instructions and therefore it is highly portable. It has data definition and data manipulation commands which are simple to use for application programmers. It has sophisticated interactive commands to query the database, modify the database and display the database schema. The system has capability to store, retrieve, modify and delete a database using both application call statements and interactive commands. The data can be integer, real, double-precision and character words. The data can be organized in the form of relations. The system has powerful relational algebra commands like INTERSECT, PROJECT and JOIN. MIDAS/R has capability to provide access to the database simultaneously for multiple users. Database security is provided through two level password system. Error recovery mechanisms are available.

### 6.2.2 Database of MIDAS/R

MIDAS/R has capability to create a number of databases. The databases can be used one at a time. The database can be either permanent or temporary. The size of a database is unlimited but depends only on the availability of disk space. Database of MIDAS/R can hold any number of relations each of which is identified by a unique name. A relation can store data of a number of attributes. An attribute value can be a single data item, a vector or a matrix. Variable length rows of a relation can be stored.



### 6.2.3 Data Definition Commands of MIDAS/R

Data definition commands are used in an application program to define a database, relations and attributes. These commands are FORTRAN subroutine call statements. These commands were not available in RIM program. The data definition commands of MIDAS/R are described in the following paragraphs.

#### Database Initialization.

CALL RDBINT

This command initializes MIDAS/R. Before using any other commands of the system, this command must be used.

#### Database Definition.

CALL RDBDFN (NAME, STAT, IERR)

NAME = Name of the database

STAT = Permanent or temporary status of the database

IERR = Error Code

A unique database can be defined using this command. A temporary database is deleted when it is closed.

#### Relation Definition.

CALL RELDFN (NAME, RNAME, NCOL, CNAME, CTYPE, IELM, JELM, KEY,  
IERR)

NAME = Name of the database

RNAME = Relation name

NCOL = Number of attribute columns

CNAME = A vector of attribute names



CTYPE = A vector of attribute type  
IELM = A vector of row size of attributes  
JELM = A vector of column size of attributes  
KEY = A vector of key attribute indicator  
IERR = Error code

A relation can be defined using this command. Relation name and attribute names must be unique in a database. A row and column intersection in a relation table can contain either a single data item, a vector or a matrix. Details of data type and layout of data in a typical relation are given in Figs. 6.2.1 and 6.2.2, respectively.

#### **Data Set Definition.**

CALL RDSDFN (NAME, DSNAME, DTYPE, IELM, JELM, IERR)

NAME = Name of the database  
DSNAME = Name of the data set  
DTYPE = Data type (see Fig. 6.2.1)  
IELM = Row size of a attribute (see Fig. 6.2.1)  
JELM = Column size of a attribute (see Fig. 6.2.1)  
IERR = Error code

A data set is defined as a collection of data belonging to same data type such as single data item, vector, or matrix. In a sense, a data set is a relation having only one attribute. Name of data set has to be unique in a database.



Description	TYPE	IELM	JELM
Integer	INT	1	1
Real	REAL	1	1
Double Precision	DOUB	1	1
Integer Vector	IVEC	n	1
Real Vector	RVEC	n	1
Double Precision Vector	DVEC	n	1
Integer Matrix	IMAT	m	n
Real Matrix	RMAT	m	n
Double Precision Matrix	DMAT	m	n
Text or Character	TEXT	n	1

NOTE: Values of IELM and JELM if 0 indicate that data is of variable length.

Figure 6.2.1 Data Type and Size of a Relation



Attribute A Key	Attribute B Type INT	Attribute C IVEC	Attribute D IMAT
1	x	x x x x	x x x x x x x x x x x x x x x x
2	x	x x x x	x x x x x x x x x x x x x x x x
..	..	..	. . . .
r	x	x x x x	x x x x x x x x x x x x x x x x

NOTE: For Attribute A IELM = 1, JELM = 1  
Attribute B IELM = 1, JELM = 4  
Attribute C IELM = 4, JELM = 4

Figure 6.2.2 Layout of Data in a Typical Relation



**Data Set Redefinition.**

```
CALL RDRDFN (NAME, DSNAME, DTYPE, IELM, JELM, IERR)
```

Arguments are same as in data set definition. This command redefines a data set using new data type, and new attribute size. Old data set definition and its data is lost.

**Data Definition Ending.**

```
CALL RDSEND (IERR)
```

IERR = Error Code

After database, relations and data sets have been defined, data definition process is ended by calling this routine. During execution of this call statement, the data definition is verified and compiled internally.

**6.2.4 Data Manipulation Commands of MIDAS/R**

Data manipulation commands open, close, store, retrieve, modify, and delete data, rename a relation or a data set, rename an attribute and copy data sets in a database. These commands were not available in the RIM program. The function and description of these commands are given in the following paragraphs.

**Open a Database.**

```
CALL RDBOPN (NAME, STAT, IERR)
```

NAME = Name of the database

STAT = Permanent or temporary status of the database

IERR = Error code



A database closed earlier can be opened using this command. A database has to be opened before any operation on the database is performed.

#### **Close a Database.**

```
CALL RDBEND (IERR)
```

IERR     = Error code

A database is closed using this command. Execution of this command transfers the system buffer data into the database and closes the file.

#### **Store Data in a Relation.**

```
CALL RDSPUT (NAME, DSNAME, KROW, UBUF, IERR)
```

NAME     = Name of the database

DSNAME   = Name of a relation

KROW     = Row number

UBUF     = User buffer which contain data

IERR     = Error code

Data can be stored into a relation from application program work area (user buffer) by using this command. Data is transferred from user buffer to the specified row of a relation. If more rows have to be stored, a FORTRAN DO loop over the row number in the application program will transfer all the required rows. More details of this command are given in the user's manual (Sreekanta Murthy and Arora, 1984).

#### **Retrieve Data from a Relation.**

```
CALL RDSGET (NAME, DSNAME, KROW, UBUF, IERR)
```



Data can be retrieved from a relation into a user buffer using this command. Requested row of a relation is transferred from a relation into user buffer. FORTRAN DO loop over the row number is necessary if more than one row has to be retrieved. Data can be retrieved in the same order as it was stored by initializing row number as zero. Data of a relation satisfying certain condition (for example, attributes having certain values) can be retrieved into user buffer. User specifies the condition on data values that must be satisfied for retrieval, by using RDSRUL command (explained later). The details for various ways of data retrieval is given in the user's manual (Sreekanta Murthy and Arora, 1984). Arguments are the same as in RDSPUT.

#### **Modify Data in a Relation.**

```
CALL RDSMOD (NAME, DSNAME, KROW, UBUF, IERR)
```

Once a database is loaded using RDSPUT command, it can be modified by calling RDSMOD. This routine modifies a row of the relation. RDSGET routine is called before calling this subroutine. A row of a relation is retrieved into user buffer and this row or a part of the row can be modified and stored back using the command. Arguments are the same as in RDSPUT.

#### **Delete Rows of a Relation.**

```
CALL PRWDEL (NAME, DSNAME, KROW, IERR)
```

Rows of a relation can be deleted using this command. This is useful in eliminating unwanted values in a relation. Arguments are the same as in RDSPUT.



**Delete a Relation.**

CALL RDSDEL (NAME, DSNAME, IERR)

This command deletes a relation from a database. Arguments are the same as in RDSPUT.

**Rename a Relation.**

CALL RDRNAM (NAME, OLDNAM, NEWNAM, IERR)

NAME = Name of a database

OLDNAM = Old name of the relation

NEWNAM = New name of the relation

IERR = Error code

An existing relation's name can be changed to a new name using the command.

**Rename an Attribute.**

CALL RRNATT (NAME, DSNAME, OLDATT, NEWATT, IERR)

NAME = Name of a database

DSNAME = Relation name

OLDATT = Old attribute name

NEWATT = New attribute name

IERR = Error code

**Copy a Relation.**

CALL RDSCPY (NAME1, NAME2, DSNAM1, DSNAM2, IERR)\*#AM2, IERR)

NAME1 = Name of a database containing data

NAME2 = Name of a database where data has to be copied

DSNAM1 = Relation name containing data



DSNAM2 = Relation name to where data has to be copied

IERR = Error code

Using this command, data from one relation can be copied to another relation. Both the database and relation must have been defined before copying the data.

### Condition Specification for Retrieval of Data.

CALL RDSRUL (NUM, ATNAM, COND, VALUE, BOOL, IERR)

NUM = Number of conditions

ATNAM = A vector of attribute names

COND = A vector of logical operator (EQ, GT, LT)

VALUE = A vector of attribute values

BOOL = A vector of Boolean operator (AND, OR)

IERR = Error code

As mentioned in RDSGET command, data values satisfying certain conditions can be retrieved. The conditions can be specified using RDSRUL command. This command must be executed before calling RDSGET routine. A maximum of ten conditions can be specified at a time. The following example, illustrates use of this command.

Condition on a relation X:

Attribute A • GT • 15.3 • AND • Attribute B • LT • 20.1

use NUM = 2; ATNAM(1) = 'A'; ATNAM(2) = 'B'

COND(1) = 'GT'; COND(2) = 'LT';

VALUE(1) = 15.3; VALUE(2) = 20.1;

BOOL(1) = 'AND'



### 6.2.5 Interactive Commands

MIDAS/R provides interactive support for creating, updating, modifying, and deleting a database. Interactive commands are general and can be used in any application. The system provides terminal prompts for the users to respond with appropriate commands. The interactive session starts with a display of MENU and requests the user to choose one of the five options: CREATE, UPDATE, QUERY, COMMAND and EXIT. The interactive session ends with an EXIT command. The detailed commands for each of these options are entered at appropriate instant. They are given in the following paragraphs.

#### Database Definition.

CREATE XXXX

Create command branches out to interactively define a new database. System responds by requesting database, relation and attribute names, and authorization access details. User can supply these data at the prompt. Details of using this command are given in user's manual (Sreekanta Murthy and Arora, 1984).

#### Loading a Database.

After a database has been successfully created, it may be loaded before ending 'create' session. For user response 'Y' for the load prompt, the list of existing relations that may be loaded are displayed. The values of attributes are entered corresponding to each attribute type.



### Querying a Database.

A database defined and loaded as specified in the previous paragraphs can be queried. Query will be in the COMMAND mode of interactive session. There are a number of query commands which allow users to query a database. They are described in the following paragraphs.

**SELECT.** Select command is used for displaying data of a relation. Options to display all or selected attributes are available. Several possible select options are given below:

```
SELECT ALL FROM [relation name]
```

```
SELECT [attribute name] FROM [relation name]
```

```
SELECT ALL FROM [relation name] WHERE [attribute name] [condition]  
[values] [AND/OR] ...
```

continuation dots indicate that upto ten conditions can be specified.

The conditions have to be one of the following

(a) [attribute name] [EQ|NE|GT|LT|LE|GE] [value]

(b) [attribute name] [EQ|NE|GT|LT|LE|GE] [attribute name]

(c) ROWS [EQ|NE|LT|LE|GE] [row number]

**LISTREL.** List command allows user to display information about relations and attributes. The following options are available in LISTREL command.

```
LISTREL
```

```
LISTREL [relation name]
```

```
LISTREL ALL
```



**CHANGE.** Data values in a relation can be changed using this command. The following options are available

CHANGE [attribute] TO [value] IN [relation name]

CHANGE [attribute] TO [VALUE] IN [relation name]

WHERE ...

**DELETE.** This command can be used to delete selected rows in a relation:

DELETE ROWS FROM [relation name] WHERE ...

**RENAME.** Attributes and relations can be renamed using this command:

RENAME [attribute] TO [attribute] In [relation name]

RENAME RELATION [relation name] TO [relation name]

**REMOVE.** A relation can be deleted from database definition using REMOVE command:

REMOVE [relation name]

**PROJECT.** This is a relational algebra command. The function of PROJECT is to create a new relation as a subset of an existing relation. The new relation is created from an existing relation by removing attributes, rows or both.

PROJECT [relation name] FROM [relation name]

USING [attribute] [ attribute] ... WHERE ...



**JOIN.** The purpose of JOIN command is to combine two relations based on specific attributes from each row. The result of JOIN command is a third relation containing all the specified attributes from both the relations.

```
JOIN [relation name] USING [attribute name]
WITH [relation name] USING [attribute name]
FORMING [relation name] WHERE ...
```

**INPUT.** This command assigns input file for MIDAS/R to read data without user interaction

```
INPUT [file name]
```

**OUTPUT.** The output of execution may be placed in a given file name. If file name is TERMINAL, then all messages and data are displayed at the terminal:

```
OUTPUT [file name]
```

**EXIT/QUIT.** The system buffer data is transfered to database files and database is closed.

**HELP.** User can obtain a description of the available commands on the terminal.

### 6.2.6 System Design of MIDAS/R

Database structure of MIDAS/R is same as the RIM database. Each database consists of three files. The first file stores relation and attribute names and their details. The second file contains the actual



data. The third file contains pointers to keyed attributes. The program is written in FORTRAN77. It consists of a number of subroutines having well defined functions. Functions of these subroutines can be generally classified into (i) command processors, (ii) input-output processor, (iii) file definition and initialization routines, (iv) memory management routines, (v) addressing and searching routines, (vi) conditional clause and rule processing routines, and (vii) security routines.

Extensions to RIM program are made to include dynamic data definition facility. The command processing routines were changed to incorporate this facility. Any new definition of a relation or an attribute during execution of a program is stored in a file for processing. Command processing routines are used to verify syntax of the relation and attribute definition. Later, the data definitions are compiled and stored in the first file of the database.

Modification of the application interface commands are made to simplify the data storage, retrieval and modification of data. The original set of conventional data manipulation commands in RIM were found to be tedious to use in applications such as finite element analysis programs. This was because, even for a simple retrieval of data at least two or three DML calls had to be made. In the MIDAS/R program, such storage and retrieval of data is made generally using one call statement.

MIDAS/R data manipulation commands use relation and database names for data manipulation operations, whereas in RIM, users are required to



assign integer numbers to refer to predefined relations of a database that are to be manipulated. The use of database and relation names in data manipulation commands reduces programming errors on the part of users which otherwise may cause reference to a wrong relation. Also, by specifying database and relation names, user is allowed to refer to a relation in another database directly without using commands for opening and closing of database files. Arguments of data manipulation commands of MIDAS/R for storage, retrieval and modification have provisions for specification of row number of a relation. The routines for storage and retrieval internally use the row numbers to transfer users data to specified locations in a relation. This provision of row number specification in the argument has facilitated in simplifying application program logic.

To improve the performance of MIDAS/R, an additional memory management interface is added. The memory management scheme allocates available memory into a number of pages of fixed size. 'Least recently used' page replacement scheme is used for replacement of pages. The page size and number of pages can be altered by changing certain parameters in the system.

#### **6.2.7 Limitations of MIDAS/R**

There are a number of limitations of the MIDAS/R program. One of the limitations is that program does not have capability to operate on a number of databases simultaneously. Secondly, the maximum size of a row in a relation cannot exceed 1024 words. If the size of a row exceeds



BD-A174 450

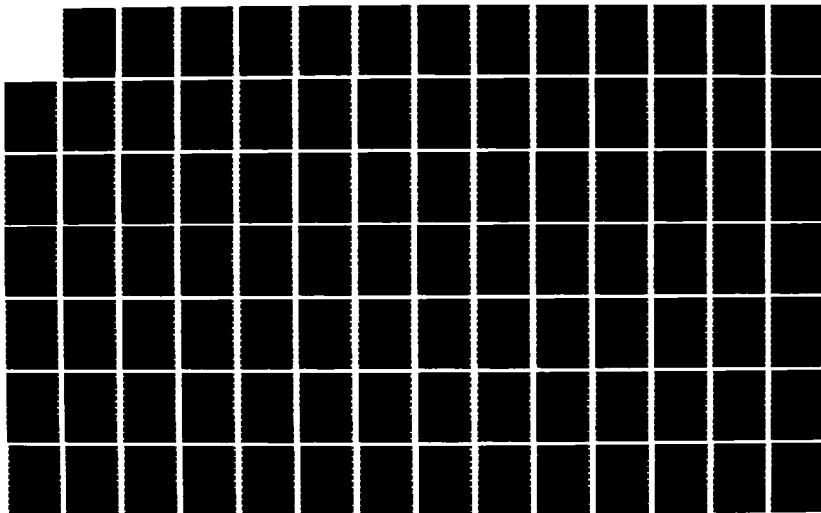
COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION USING A  
DATABASE MANAGEMENT (U) IOWA UNIV IOWA CITY OPTIMAL  
DESIGN LAB T SREEKANTAMURTHY ET AL 30 SEP 86

3/4

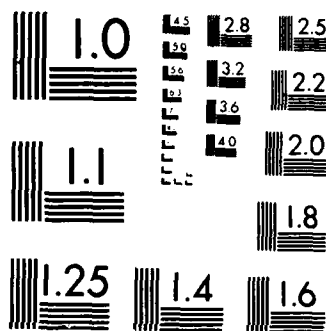
UNCLASSIFIED

ODL-85-17 AFOSR-TR-86-2069 AFFSR-82-0322 F/G 18/3

NL







MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A



this limit, user should split the row suitably and operate on portions of a row at a time. Also, the number of attributes in a relation cannot exceed 20. The memory management scheme has fixed block size. User has no control over the block size to tune it according to the relation size. At present only five relations can be operated at a time in the system buffer as only five pointers to current relations are maintained by the DBMS. Detailed evaluation of efficiency of MIDAS/R is given in Chapter IX.

### 6.3 Description of MIDAS/N

MIDAS/N is a database management system (Shyy, 1985) to support data organization of numerical computations. MIDAS/N is implemented based on numerical data model. It has data definition and data manipulation commands to define large order matrix data and manipulate them. Large matrices such as rectangular, square, upper triangular, lower triangular and hyper matrices can be defined in the database. Matrix data can be arranged in row, column, or submatrix order. Data can be short integer, long integer, real, double-precision and character types. Data manipulation commands of MIDAS/N can store, retrieve, modify and delete matrices. Data can be accessed in row column or submatrix order. Also individual data elements of a matrix can be accessed. Database security is provided through a password access. Error recovery mechanisms are available.

MIDAS/N has capability to create a number of databases, upto a maximum of 20 in the current implementation. These databases can be



accessed simultaneously. The databases can be permanent or temporary. A database can store data of a number of matrices, upto a maximum of 20. The size of a database and a matrix is unlimited, but only depends on the availability of disk space. Databases can be organized at a number of hierarchical levels and can be accessed using a path name. Databases and matrices are identified by a unique name.

In MIDAS/N, the available memory (called the buffer) is divided into pages. Data set is also divided into pages of the same size when it is defined or transferred into the memory buffer. MIDAS/N handles all access requests and transactions against data sets. Each transaction is in terms of pages. Once part of a data set is requested, some pages must be assigned to it. When there is no free page, some page must be freed. Page replacement strategy decides the page to be freed. MIDAS/N chooses the 'least recently used' page for replacement. For handling transactions, information about data sets and page usage must be kept and managed. Information about data set is stored in the 'data set information' table. Information about page usage is stored in the 'memory buffer information' table. Page size and number of pages can be altered by changing certain parameters in the system.

The MIDAS/N system maintains an index table to provide address of stored records. Index table provides a pointer to the data definition block which contains details of a data set such as name, type, order, and size. Data definition block is stored at the beginning of actual data in a file. Data set is mapped on to physical storage space in a linear address sequence.



MIDAS/N has been evaluated for solving equations using skyline storage scheme (Shyy, 1985). The efficiency of MIDAS/N will be compared with that of MIDAS/R in solving large equations. The results are given in Chapter 9.



## CHAPTER 7

### A DESIGN OF DATABASE FOR STRUCTURAL ANALYSIS AND OPTIMIZATION DATA

#### 7.1 Introductory Remarks

A database is designed for finite element analysis and structural optimization computation and its details are given in this chapter. Methodology developed in Chapter 4 is used in designing the database. The database design is carried out in three phases. In the first phase, a conceptual database is designed to provide a theoretical basis for organizing finite element analysis and structural optimization data. The design of the conceptual data model is given in Section 7.3. In the second phase, a generalized internal model is designed which is used later to implement a database for structural design optimization program discussed in the next chapter. The design of the internal model is given in Section 7.4. In Section 7.5, some relations are suggested to form an external data model. Finally, the methodology used in designing the database is evaluated.

#### 7.2 Identification of Data Used in Finite Element Analysis and Structural Design Optimization

Database design is initiated by identifying data required for finite element analysis and optimal structural design. In Sections 2.2 and 2.5, we have already described in detail the data used in structural



analysis and optimization. Here, they are identified again from the database design point of view. For the design of a conceptual data model, we need to identify entities, attributes, domains and entity keys. In the following subsections, a list of entities, entity keys, and domains of data are given.

### **7.2.1 A List of Entities of Analysis and Design Data**

The following entities are identified from the data used in finite element analysis and optimal design computations (see Sections 2.2 and 2.5):

1. Structure
2. Substructure
3. Element
4. Node
5. Material
6. Element type
7. Material type
8. Cross-section type
9. Degree of freedom
10. Load case
11. Design Group
12. Design Variable
13. Global constraint
14. Element constraint
15. Node constraint



16. Structure constraint

17. Mode

18. Matrix name

19. Vector name

20. Constraint type

21. Load type

22. Member

The following entity keys are identified. Note that an entity key uniquely identifies an entity.

1. Structure number (or name)	S#
2. Substructure number	SUBSTR#
3. Element number	E#
4. Node number	N#
5. Material number	M#
6. Element type number (or name)	ELTYP
7. Material type number (or name)	MTTYP
8. Cross-section type number (or name)	CSTYP
9. Degree of freedom number	DOF#
10. Load case number	LC#
11. Design group number	DG#
12. Design variable number	D#
13. Global constraint number	GC#
14. Element constraint number	EC#
15. Nodal constraint number	NC#



16. Structure constraint number	SC#
17. Mode number	MODE#
18. Matrix name	MN
19. Vector name	VN
20. Constraint type	CTTYP
21. Load type	LDTYP
22. Member	ME#

### 7.2.2 A List of Domains of Analysis and Design Data

The following domains are identified:

- |  |        |
|--|--------|
| 1. Assembled structure level matrices; e.g., $K, M, C$   | STRMAT |
| 2. Assembled structure level vectors; e.g., $P, U$   | STRVEC |
| 3. Structure level parameters, indices, flags, etc.<br>e.g., No. of substructures, No. of load cases | STRPAR |
| 4. Assembled substructure level matrices<br>e.g., $K_{ij}, K_{bi}, K_{bb}$                           | SUBMAT |
| 5. Assembled substructure level vectors; e.g., $U_i, U_b$  | SUBVEC |
| 6. Substructure level parameters, indices, flags, etc.<br>e.g., No. of elements, No. of nodes        | SUBPAR |
| 7. Element level matrices; e.g., $K_e, M_e$  | ELMAT  |
| 8. Element level vectors e.g., $\sigma_e, \epsilon_e, \frac{\partial K_e}{\partial b} U_e$           | ELVEC  |
| 9. Element level parameters, indices, flags, etc.<br>e.g., Nonlinearity index, damage flag           | ELPAR  |
| 10. Node related vectors; e.g., boundary condition codes   | NODVEC |



- |   |        |
|---|--------|
| 11. Node related parameters, indices, flags, etc.<br>e.g., coordinate, temperature  | NODPAR |
| 12. Material property tables<br>e.g., Nonlinear material property tables  | MATTAB |
| 13. Material property vectors<br>e.g., stress limit vector  | MATVEC |
| 14. Material property parameters<br>e.g., Density, Young's Modulus  | MATPAR |
| 15. Vectors associated with degree of freedom<br>e.g., velocity at specified set of time points                           | DOFVEC |
| 16. Parameters, indices, flags associated with<br>degrees of freedom; e.g., Prescribed displacement,<br>constraint values | DOFPAR |
| 17. Vectors associated with a design group<br>e.g., Vector of element numbers   | DESVEC |
| 18. Parameters associated with a design group<br>e.g., fixed design flag  | DESPAR |
| 19. Vectors associated @ th a design variable<br>e.g., upper bound, lower bound, current value                            | DEVVEC |
| 20. Parameters associated with a design variable<br>e.g., fixed design index  | DEVPAR |
| 21. Matrices associated with global constraints; e.g., $\frac{d\psi}{db}$   | GLCMAT |
| 22. Vectors associated with global constraints; e.g., $\psi$  | GLCVEC |
| 23. Parameters, indices, flags associated with<br>global constraint; e.g., active constraint number                       | GLCPAR |



- |     |   |        |
|-----|---|--------|
| 24. | Matrices associated with element level constraints<br>e.g., $\frac{\partial K_e}{\partial b}$   | ELCMAT |
| 25. | Vectors associated with element level constraints<br>e.g., $\frac{\partial \sigma_e}{\partial z}$                                       | ELCVEC |
| 26. | Parameters associated with element level constraints<br>e.g., Normalized element constraint value,<br>$\psi = \sigma_e / \sigma_a - 1$  | ELCPAR |
| 27. | Vectors associated with node constraints<br>e.g., $\psi = z_j / z_j^a - 1 \quad j = 1, \text{ no. of dof/node}$                         | NDCVEC |
| 28. | Vectors associated with structure level constraints<br>e.g., $\frac{\partial K_z}{\partial b} - \zeta \frac{\partial M}{\partial b} z)$ | STCVEC |
| 29. | Parameters associated with structure level constraints<br>e.g., $\psi = \zeta / \zeta_0 - 1$  | STCPAR |
| 30. | Vectors associated with frequency/buckling; e.g., $y$   | MODVEC |
| 31. | Parameters associated with frequency/buckling<br>e.g., $\lambda$  | MODPAR |
| 32. | Names in characters<br>e.g., matrix name, vector name, element name   | NAME   |
| 33. | Numbers in integers values<br>e.g., element numbers, node numbers   | INT    |
| 34. | Numbers in real values; density, temperature  | REAL   |
| 35. | Numbers in double precision values; e.g., frequency   | DOUB   |



### 7.3 Design of a Conceptual Data Model

Following the methodology, a rough conceptual data model is designed by associating entities with domains. Details of this model are given in Section 7.3.1. Later, this model is refined by introducing additional relations derived from transitive closure. Details of this step are given in Section 7.3.2. From the obtained set of elementary relations a set of semantically meaningful relations are selected. A refined conceptual data model is finally obtained by removing redundant elementary relations.

#### **7.3.1 Elementary Relations and Diagraph Representation**

By associating the entities given in Section 7.2.1 with the domains listed in Section 7.2.2, we get a set of relations. For example, by associating entity structure with domains STRMAT, STRVEC, and STRPAR and with entities MN and VC, we get the relation:

R1 (S, MN, VN, STRMAT, STRVEC, STRPAR)

Note that S, MN, VN, STRMAT, STRVEC, and STRPAR are attributes of this relation. Entities MN and VN are assigned with entity S to identify the actual matrix and vector associated with STRMAT and STRVEC. It is possible to further derive several attributes from domains STRMAT, STRVEC and STRPAR to provide role identification in relation R1. For example, attributes K and M can be derived to provide role identification for domain STRMAT. But, at this stage of database design, such details are omitted to maintain generality of the conceptual data model. Details are introduced at the actual



implementation stage of database design. Therefore, STRMAT is considered here as a generalized attribute which actually represents  $K$ ,  $M$ ,  $C$ , etc.

Reducing this relation to several elementary relation we get

STRUCTURE - MATRIX ( $S\#$ ,  $MN$ , STRMAT)

STRUCTURE - VECTOR ( $S\#$ ,  $VN$ , STRVEC)

STRUCTURE - PARAMETER ( $S\#$ , STRPAR)

Here, the attribute STRMAT is fully-functionally dependent on  $S\#$  and  $MN$ .

A diagraph representation of the association between entity structure and various attributes (for example - assembled stiffness matrix, mass matrix) derived from domain STRMAT and STRVEC is shown in Fig. 7.3.1. Similarly other elementary relations are derived and they are listed in Fig. 7.3.2. The list also includes the relations between entities themselves. This list of elementary relations forms a rough conceptual data model to represent finite element analysis and structural design optimization data.

### 7.3.2 Deriving Additional Relations

An initial connectivity matrix representing the elementary relations of Fig. 7.3.2 is made. This is shown in Fig. 7.3.3. Using the algorithm of Appendix 1, additional relations are derived. The algorithm uses the initial connectivity matrix and produces a final connectivity matrix shown in Fig. 7.3.4. From this matrix, additional elementary relations formed are identified. Several hundred new relations are obtained as indicated in Fig. 7.3.5. These additional



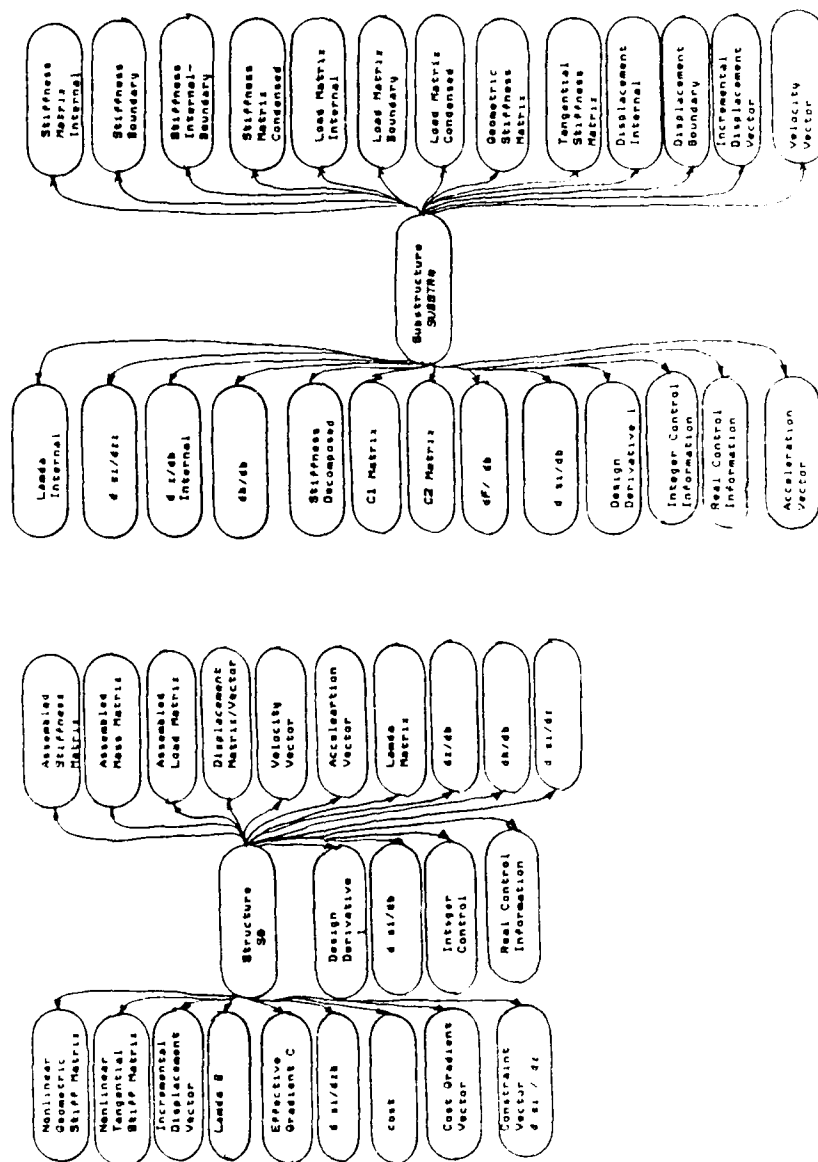


Figure 7.3.1 Diagram Representation of Association Between Entities and Attributes



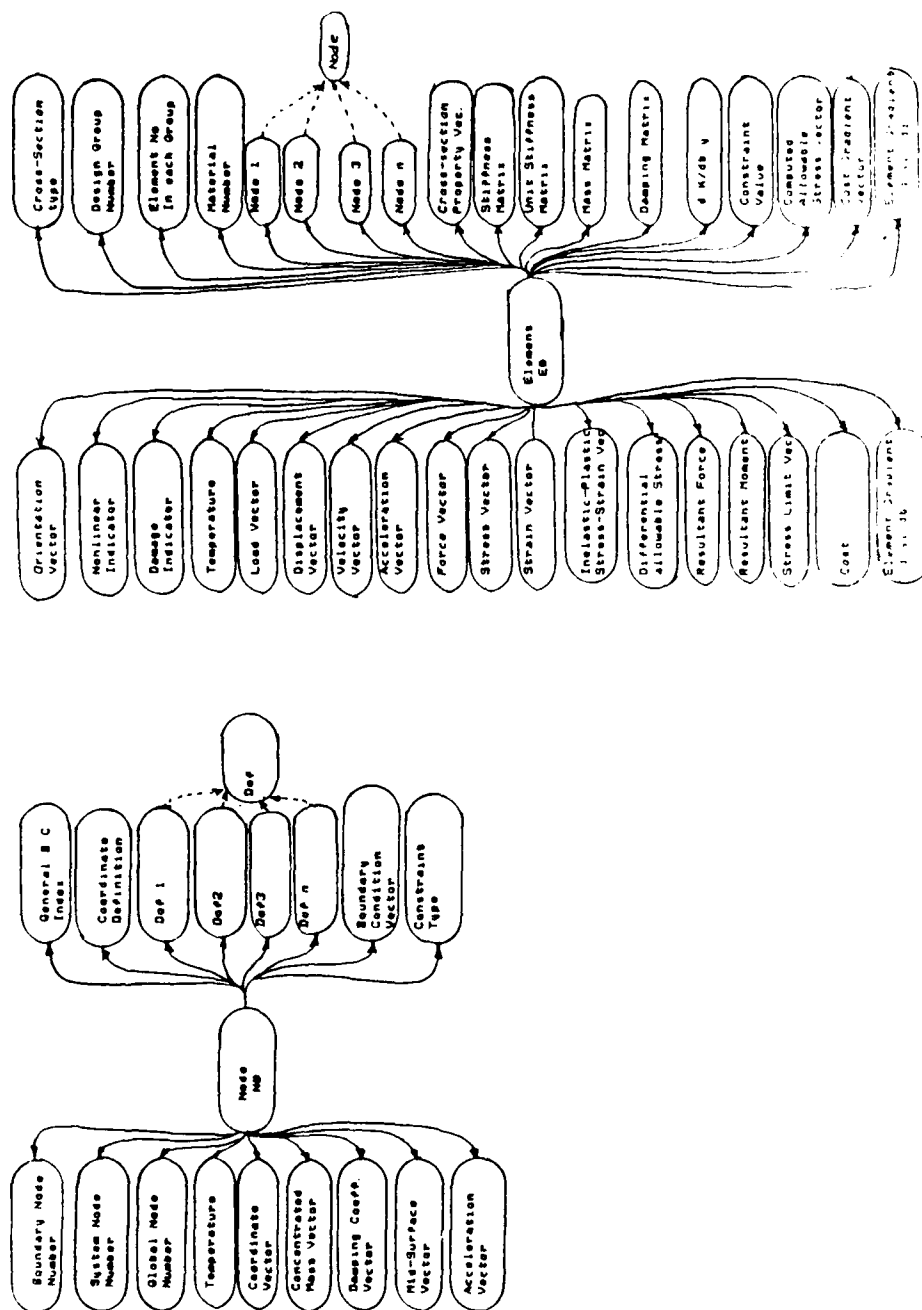


Figure 7.3.1 (Continued)



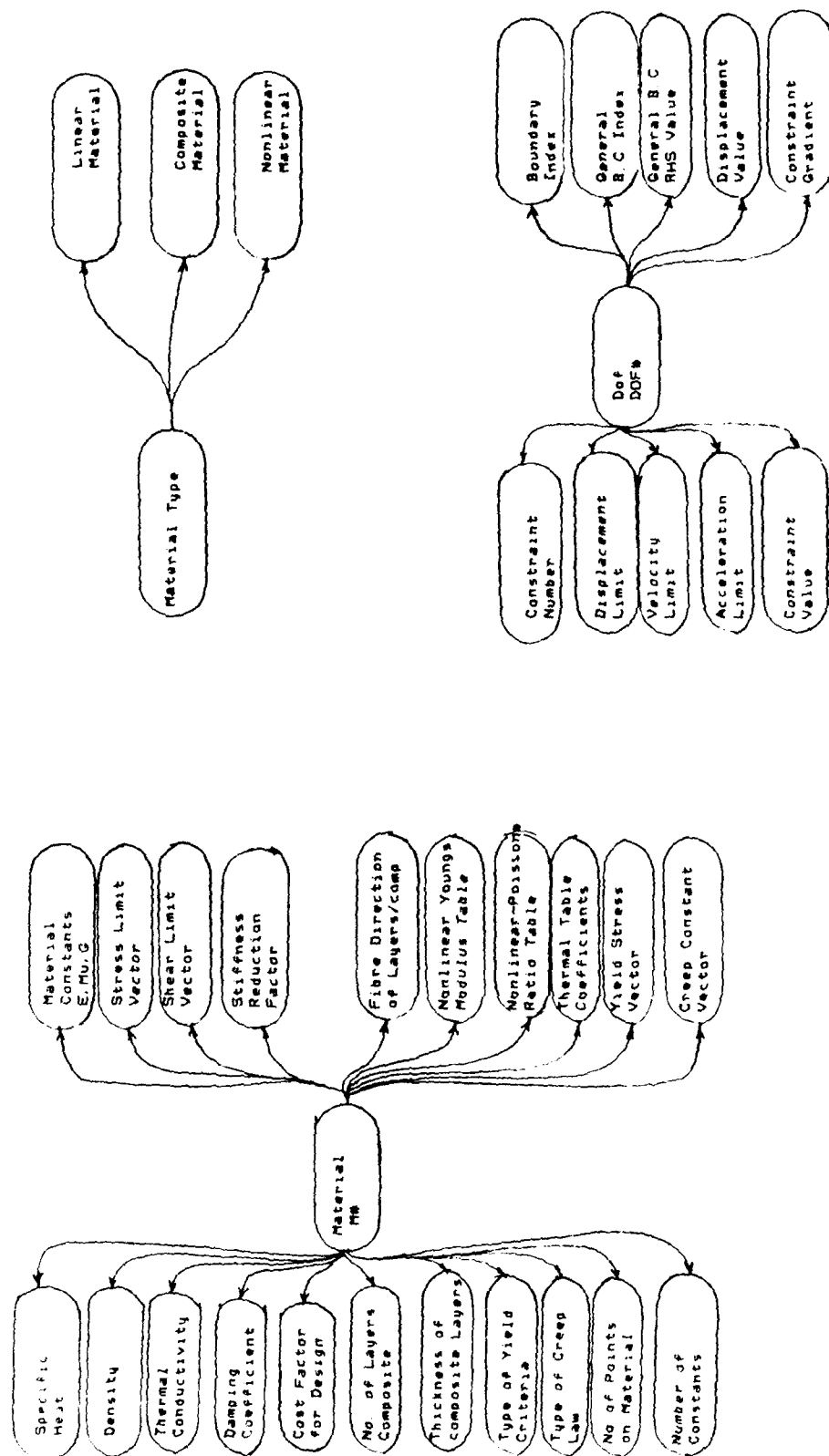


Figure 7.3.1 (Continued)



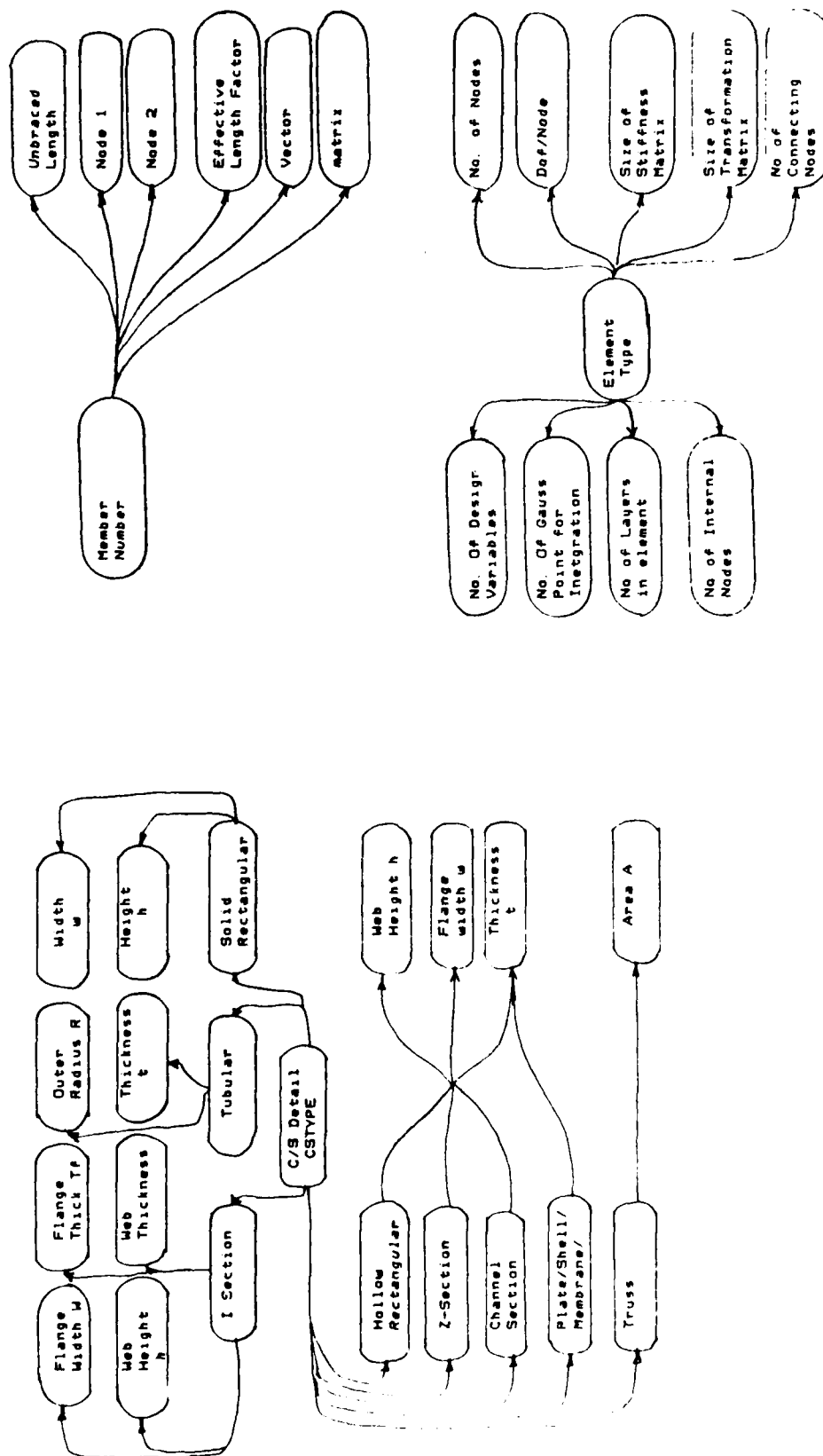


Figure 7.3.1 (Continued)



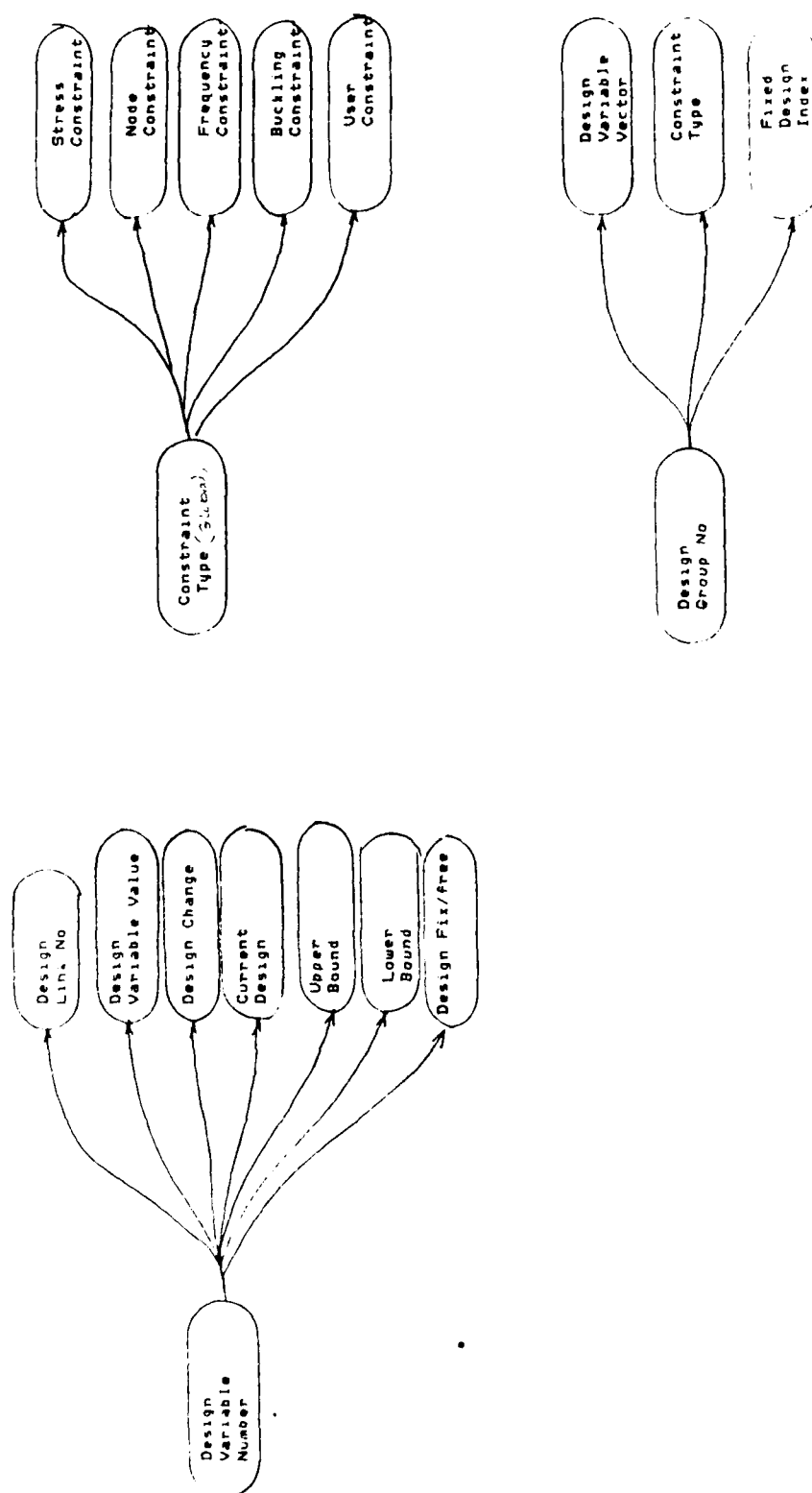


Figure 7.3.1 (Continued)



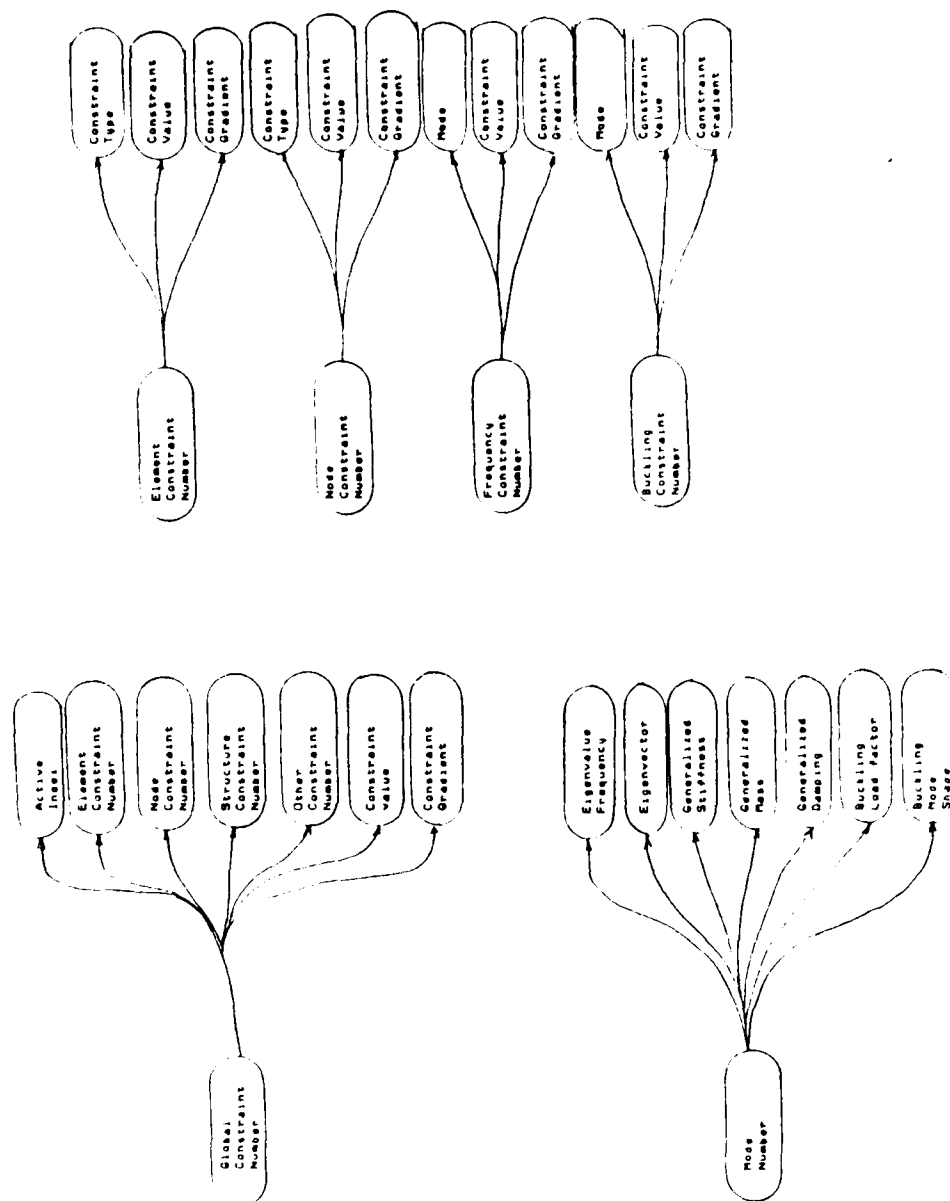


Figure 7.3.1 (Continued)



## INITIAL ELEMENTARY RELATIONS

ER 1	( S#, MN	, S#	)	ER 46	( DOF#	, DOFPAR	)
ER 2	( S#, MN	, STRMAT	)	ER 47	( LC#	, LDTYP	)
ER 3	( S#, VN	, S#	)	ER 48	( DO#, MN	, DESMAT	)
ER 4	( S#, VN	, STRVEC	)	ER 49	( DO#, VN	, DESVEC	)
ER 5	( S#	, STRPAR	)	ER 50	( DO#	, DESPAR	)
ER 6	( S#	, SUBSTR#	)	ER 51	( DO#, VN	, DEVVEC	)
ER 7	( S#	, GC#	)	ER 52	( DO#	, DEVPAR	)
ER 8	( S#	, MODE#	)	ER 53	( GC#, MN	, GLCMAT	)
ER 9	( SUBSTR#, MN	, SUBSTR#	)	ER 54	( GC#, VN	, GLCVEC	)
ER 10	( SUBSTR#, MN	, SUBMAT	)	ER 55	( GC#	, GLCPAR	)
ER 11	( SUBSTR#, VN	, SUBSTR#	)	ER 56	( GC#	, EC#	)
ER 12	( SUBSTR#, VN	, SUBVEC	)	ER 57	( GC#	, NC#, VN	)
ER 13	( SUBSTR#	, SUBPAR	)	ER 58	( GC#	, SC#	)
ER 14	( SUBSTR#	, E#, ELTYP	)	ER 59	( GC#	, CTTYP	)
ER 15	( SUBSTR#	, D#	)	ER 60	( EC#, MN	, ECHMAT	)
ER 16	( SUBSTR#	, SC#	)	ER 61	( EC#, VN	, ECVEC	)
ER 17	( SUBSTR#	, MODE#	)	ER 62	( EC#	, ECPAR	)
ER 18	( SUBSTR#	, ME#	)	ER 63	( NC#, VN	, NDCVEC	)
ER 19	( E#, ELTYP, MN	, E#, ELTYP	)	ER 64	( SC#, VN	, STCVEC	)
ER 20	( E#, ELTYP, MN	, ELMAT	)	ER 65	( SC#	, STCPAR	)
ER 21	( E#, ELTYP, VN	, E#, ELTYP	)	ER 66	( MODE#, VC	, MODVEC	)
ER 22	( E#, ELTYP, VN	, ELVEC	)	ER 67	( MODE#	, MODPAR	)
ER 23	( E#, ELTYP	, ELPAR	)	ER 68	( ME#	, E#, ELTYP	)
ER 24	( E#, ELTYP	, N#	)	ER 69	( S#, LC#	, S#	)
ER 25	( E#, ELTYP	, M#, MATTYP	)	ER 70	( SUBSTR, LC#	, SUBSTR#	)
ER 26	( E#, ELTYP	, ELTYP	)	ER 71	( E#, LC#	, E#, ELTYP	)
ER 27	( E#, ELTYP	, MATTYP	)	ER 72	( N#, LC#	, N#	)
ER 28	( E#, ELTYP	, CSTYPE	)	ER 73	( GC#, LC#	, GC#	)
ER 29	( E#, ELTYP	, DO#	)	ER 74	( DOF#, LC#	, DOF#	)
ER 30	( E#, ELTYP	, EC#	)				
ER 31	( E#, ELTYP	, CTTYP	)				
ER 32	( N#, VN	, NODVEC	)				
ER 33	( N#	, NODPAR	)				
ER 34	( N#	, DOF#	)				
ER 35	( N#	, NC#, VN	)				
ER 36	( N#	, CTTYP	)				
ER 37	( M#, MATTYP, MN	, M#, MATTYP	)				
ER 38	( M#, MATTYP, MN	, MATTAB	)				
ER 39	( M#, MATTYP, VN	, M#, MATTYP	)				
ER 40	( M#, MATTYP, VN	, MATVEC	)				
ER 41	( M#, MATTYP	, MATPAR	)				
ER 42	( M#, MATTYP	, MATTYP	)				
ER 43	( ELTYP	, ELPAR	)				
ER 44	( CSTYPE	, ELPAR	)				
ER 45	( DOF#, VN	, DOFVEC	)				

Figure 7.3.2 Initial List of Elementary Relations







100

**Figure 7.3.4 Final Connectivity Matrix**



## TRANSITIVELY CLOSED RELATIONS

ER 1	( S#, MN	, S#	)	ER 65	( S#	, MATTYP	)
ER 2	( S#, MN	, STRMAT	)	ER 66	( S#	, CSTYPE	)
ER 3	( S#, MN	, STRPAR	)	ER 67	( S#	, DG#	)
ER 4	( S#, MN	, SUBSTR#	)	ER 68	( S#	, DESPAR	)
ER 5	( S#, MN	, SUBPAR	)	ER 69	( S#	, D#	)
ER 6	( S#, MN	, E#, ELTYP	)	ER 70	( S#	, DEVPAR	)
ER 7	( S#, MN	, ELPAR	)	ER 71	( S#	, QC#	)
ER 8	( S#, MN	, M#, MATTYP	)	ER 72	( S#	, QLCPAR	)
ER 9	( S#, MN	, MATPAR	)	ER 73	( S#	, EC#	)
ER 10	( S#, MN	, SLTYP	)	ER 74	( S#	, ECPAR	)
ER 11	( S#, MN	, MATTYP	)	ER 75	( S#	, NC#, VN	)
ER 12	( S#, MN	, CSTYPE	)	ER 76	( S#	, NDCVEC	)
ER 13	( S#, MN	, DG#	)	ER 77	( S#	, SC#	)
ER 14	( S#, MN	, DESPAR	)	ER 78	( S#	, STCPAR	)
ER 15	( S#, MN	, D#	)	ER 79	( S#	, MODE#	)
ER 16	( S#, MN	, DEVPAR	)	ER 80	( S#	, MODPAR	)
ER 17	( S#, MN	, QC#	)	ER 81	( S#	, CTTYP	)
ER 18	( S#, MN	, QLCPAR	)	ER 82	( S#	, ME#	)
ER 19	( S#, MN	, EC#	)	ER 83	( SUBSTR#, MN	, SUBSTR#	)
ER 20	( S#, MN	, ECPAR	)	ER 84	( SUBSTR#, MN	, SUBMAT	)
ER 21	( S#, MN	, NC#, VN	)	ER 85	( SUBSTR#, MN	, SUBPAR	)
ER 22	( S#, MN	, NDCVEC	)	ER 86	( SUBSTR#, MN	, E#, ELTYP	)
ER 23	( S#, MN	, SC#	)	ER 87	( SUBSTR#, MN	, ELPAR	)
ER 24	( S#, MN	, STCPAR	)	ER 88	( SUBSTR#, MN	, M#, MATTYP	)
ER 25	( S#, MN	, MODE#	)	ER 89	( SUBSTR#, MN	, MATPAR	)
ER 26	( S#, MN	, MODPAR	)	ER 90	( SUBSTR#, MN	, ELTYP	)
ER 27	( S#, MN	, CTTYP	)	ER 91	( SUBSTR#, MN	, MATTYP	)
ER 28	( S#, MN	, ME#	)	ER 92	( SUBSTR#, MN	, CSTYPE	)
ER 29	( S#, VN	, S#	)	ER 93	( SUBSTR#, MN	, DG#	)
ER 30	( S#, VN	, STRVEC	)	ER 94	( SUBSTR#, MN	, DESPAR	)
ER 31	( S#, VN	, STRPAR	)	ER 95	( SUBSTR#, MN	, D#	)
ER 32	( S#, VN	, SUBSTR#	)	ER 96	( SUBSTR#, MN	, DEVPAR	)
ER 33	( S#, VN	, SUBPAR	)	ER 97	( SUBSTR#, MN	, EC#	)
ER 34	( S#, VN	, E#, ELTYP	)	ER 98	( SUBSTR#, MN	, ECPAR	)
ER 35	( S#, VN	, ELPAR	)	ER 99	( SUBSTR#, MN	, SC#	)
ER 36	( S#, VN	, M#, MATTYP	)	ER100	( SUBSTR#, MN	, STCPAR	)
ER 37	( S#, VN	, MATPAR	)	ER101	( SUBSTR#, MN	, MODE#	)
ER 38	( S#, VN	, ELTYP	)	ER102	( SUBSTR#, MN	, MODPAR	)
ER 39	( S#, VN	, MATTYP	)	ER103	( SUBSTR#, MN	, CTTYP	)
ER 40	( S#, VN	, CSTYPE	)	ER104	( SUBSTR#, MN	, ME#	)
ER 41	( S#, VN	, DG#	)	ER105	( SUBSTR#, VN	, SUBSTR#	)
ER 42	( S#, VN	, DESPAR	)	ER106	( SUBSTR#, VN	, SUBVEC	)
ER 43	( S#, VN	, D#	)	ER107	( SUBSTR#, VN	, SUBPAR	)
ER 44	( S#, VN	, DEVPAR	)	ER108	( SUBSTR#, VN	, E#, ELTYP	)
ER 45	( S#, VN	, QC#	)	ER109	( SUBSTR#, VN	, ELPAR	)
ER 46	( S#, VN	, QLCPAR	)	ER110	( SUBSTR#, VN	, M#, MATTYP	)
ER 47	( S#, VN	, EC#	)	ER111	( SUBSTR#, VN	, MATPAR	)
ER 48	( S#, VN	, ECPAR	)	ER112	( SUBSTR#, VN	, ELTYP	)
ER 49	( S#, VN	, NC#, VN	)	ER113	( SUBSTR#, VN	, MATTYP	)
ER 50	( S#, VN	, NDCVEC	)	ER114	( SUBSTR#, VN	, CSTYPE	)
ER 51	( S#, VN	, SC#	)	ER115	( SUBSTR#, VN	, DG#	)
ER 52	( S#, VN	, STCPAR	)	ER116	( SUBSTR#, VN	, DESPAR	)
ER 53	( S#, VN	, MODE#	)	ER117	( SUBSTR#, VN	, D#	)
ER 54	( S#, VN	, MODPAR	)	ER118	( SUBSTR#, VN	, DEVPAR	)
ER 55	( S#, VN	, CTTYP	)	ER119	( SUBSTR#, VN	, EC#	)
ER 56	( S#, VN	, ME#	)	ER120	( SUBSTR#, VN	, ECPAR	)
ER 57	( S#	, STRPAR	)	ER121	( SUBSTR#, VN	, SC#	)
ER 58	( S#	, SUBSTR#	)	ER122	( SUBSTR#, VN	, STCPAR	)
ER 59	( S#	, SUBPAR	)	ER123	( SUBSTR#, VN	, MODE#	)
ER 60	( S#	, E#, ELTYP	)	ER124	( SUBSTR#, VN	, MODPAR	)
ER 61	( S#	, ELPAR	)	ER125	( SUBSTR#, VN	, CTTYP	)
ER 62	( S#	, M#, MATTYP	)	ER126	( SUBSTR#, VN	, ME#	)
ER 63	( S#	, MATPAR	)	ER127	( SUBSTR#	, SUBPAR	)
ER 64	( S#	, ELTYP	)	ER128	( SUBSTR#	, E#, ELTYP	)
				ER129	( SUBSTR#	, ELPAR	)
				ER130	( SUBSTR#	, M#, MATTYP	)

Figure 7.3.5 Derived Elementary Relations



ER131	(	SUBSTR#	,	MATPAR	)	ER197	(	M#, MATTYP, VN, MATPAR	)	
ER132	(	SUBSTR#	,	ELTYP	)	ER198	(	M#, MATTYP, VN, MATTYP	)	
ER133	(	SUBSTR#	,	MATTYP	)	ER199	(	M#, MATTYP	, MATPAR	)
ER134	(	SUBSTR#	,	CSTYPE	)	ER200	(	M#, MATTYP	, MATTYP	)
ER135	(	SUBSTR#	,	DG#	)	ER201	(	ELTYP	, ELPAR	)
ER136	(	SUBSTR#	,	DESPAR	)	ER202	(	CSTYPE	, ELPAR	)
ER137	(	SUBSTR#	,	D#	)	ER203	(	DOF#, VN	, DOFVEC	)
ER138	(	SUBSTR#	,	DEVPAR	)	ER204	(	DOF#	, DOFPAR	)
ER139	(	SUBSTR#	,	EC#	)	ER205	(	LC#	, LDTYP	)
ER140	(	SUBSTR#	,	ECPAR	)	ER206	(	DG#, MN	, DESMAT	)
ER141	(	SUBSTR#	,	SC#	)	ER207	(	DG#, VN	, DESVEC	)
ER142	(	SUBSTR#	,	STCPAR	)	ER208	(	DG#	, DESPAR	)
ER143	(	SUBSTR#	,	MODE#	)	ER209	(	D#, VN	, DEVVEC	)
ER144	(	SUBSTR#	,	MODPAR	)	ER210	(	D#	, DEVPAR	)
ER145	(	SUBSTR#	,	CTTYP	)	ER211	(	QC#, MN	, GLCMAT	)
ER146	(	SUBSTR#	,	ME#	)	ER212	(	QC#, VN	, GLCVEC	)
ER147	(	E#, ELTYP, MN	,	E#, ELTYP	)	ER213	(	QC#	, GLCPAR	)
ER148	(	E#, ELTYP, MN	,	ELMAT	)	ER214	(	QC#	, EC#	)
ER149	(	E#, ELTYP, MN	,	ELPAR	)	ER215	(	QC#	, ECPAR	)
ER150	(	E#, ELTYP, MN	,	M#, MATTYP	)	ER216	(	QC#	, NC#, VN	)
ER151	(	E#, ELTYP, MN	,	MATPAR	)	ER217	(	QC#	, NDCVEC	)
ER152	(	E#, ELTYP, MN	,	ELTYP	)	ER218	(	QC#	, SC#	)
ER153	(	E#, ELTYP, MN	,	MATTYP	)	ER219	(	QC#	, STCPAR	)
ER154	(	E#, ELTYP, MN	,	CSTYPE	)	ER220	(	QC#	, CTTYP	)
ER155	(	E#, ELTYP, MN	,	DG#	)	ER221	(	EC#, MN	, ECMAT	)
ER156	(	E#, ELTYP, MN	,	DESPAR	)	ER222	(	EC#, VN	, ECVEC	)
ER157	(	E#, ELTYP, MN	,	EC#	)	ER223	(	EC#	, ECPAR	)
ER158	(	E#, ELTYP, MN	,	ECPAR	)	ER224	(	NC#, VN	, NDCVEC	)
ER159	(	E#, ELTYP, MN	,	CTTYP	)	ER225	(	SC#, VN	, STCVEC	)
ER160	(	E#, ELTYP, VN	,	E#, ELTYP	)	ER226	(	SC#	, STCPAR	)
ER161	(	E#, ELTYP, VN	,	ELVEC	)	ER227	(	MODE#, VC	, JDDVEC	)
ER162	(	E#, ELTYP, VN	,	ELPAR	)	ER228	(	MODE#	, MODPAR	)
ER163	(	E#, ELTYP, VN	,	M#, MATTYP	)	ER229	(	ME#	, E#, ELTYP	)
ER164	(	E#, ELTYP, VN	,	MATPAR	)	ER230	(	ME#	, ELPAR	)
ER165	(	E#, ELTYP, VN	,	ELTYP	)	ER231	(	ME#	, M#, MATTYP	)
ER166	(	E#, ELTYP, VN	,	MATTYP	)	ER232	(	ME#	, MATPAR	)
ER167	(	E#, ELTYP, VN	,	CSTYPE	)	ER233	(	ME#	, ELTYP	)
ER168	(	E#, ELTYP, VN	,	DG#	)	ER234	(	ME#	, MATTYP	)
ER169	(	E#, ELTYP, VN	,	DESPAR	)	ER235	(	ME#	, CSTYPE	)
ER170	(	E#, ELTYP, VN	,	EC#	)	ER236	(	ME#	, DG#	)
ER171	(	E#, ELTYP, VN	,	ECPAR	)	ER237	(	ME#	, DESPAR	)
ER172	(	E#, ELTYP, VN	,	CTTYP	)	ER238	(	ME#	, EC#	)
ER173	(	E#, ELTYP	,	ELPAR	)	ER239	(	ME#	, ECPAR	)
ER174	(	E#, ELTYP	,	M#, MATTYP	)	ER240	(	ME#	, CTTYP	)
ER175	(	E#, ELTYP	,	MATPAR	)	ER241	(	S#, LC#	, S#	)
ER176	(	E#, ELTYP	,	ELTYP	)	ER242	(	S#, LC#	, STRPAR	)
ER177	(	E#, ELTYP	,	MATTYP	)	ER243	(	S#, LC#	, SUBSTR#	)
ER178	(	E#, ELTYP	,	CSTYPE	)	ER244	(	S#, LC#	, SUBPAR	)
ER179	(	E#, ELTYP	,	DG#	)	ER245	(	S#, LC#	, E#, ELTYP	)
ER180	(	E#, ELTYP	,	DESPAR	)	ER246	(	S#, LC#	, ELPAR	)
ER181	(	E#, ELTYP	,	EC#	)	ER247	(	S#, LC#	, M#, MATTYP	)
ER182	(	E#, ELTYP	,	ECPAR	)	ER248	(	S#, LC#	, MATPAR	)
ER183	(	E#, ELTYP	,	CTTYP	)	ER249	(	S#, LC#	, ELTYP	)
ER184	(	N#, VN	,	NODVEC	)	ER250	(	S#, LC#	, MATTYP	)
ER185	(	N#	,	NODPAR	)	ER251	(	S#, LC#	, CSTYPE	)
ER186	(	N#	,	DOF#	)	ER252	(	S#, LC#	, DG#	)
ER187	(	N#	,	DOFPAR	)	ER253	(	S#, LC#	, DESPAR	)
ER188	(	N#	,	NC#, VN	)	ER254	(	S#, LC#	, D#	)
ER189	(	N#	,	NDCVEC	)	ER255	(	S#, LC#	, DEVPAR	)
ER190	(	N#	,	CTTYP	)	ER256	(	S#, LC#	, GC#	)
ER191	(	M#, MATTYP, MN, M#, MATTYP	)			ER257	(	S#, LC#	, GLCPAR	)
ER192	(	M#, MATTYP, MN, MATTAB	,			ER258	(	S#, LC#	, EC#	)
ER193	(	M#, MATTYP, MN, MATPAR	)			ER259	(	S#, LC#	, ECPAR	)
ER194	(	M#, MATTYP, MN, MATTYP	)			ER260	(	S#, LC#	, NC#, VN	)
ER195	(	M#, MATTYP, VN, M#, MATTYP	)			ER261	(	S#, LC#	, NDCVEC	)
ER196	(	M#, MATTYP, VN, MATVEC	)			ER262	(	S#, LC#	, SC#	)

Figure 7.3.5 (Continued)



ER263	( S#, LC#	, STCPAR	)
ER264	( S#, LC#	, MODE#	)
ER265	( S#, LC#	, MODPAR	)
ER266	( S#, LC#	, CTTYP	)
ER267	( S#, LC#	, ME#	)
ER268	( SUBSTR, LC#	, SUBSTR#	)
ER269	( SUBSTR, LC#	, SUBPAR	)
ER270	( SUBSTR, LC#	, E#, ELTYP	)
ER271	( SUBSTR, LC#	, ELPAR	)
ER272	( SUBSTR, LC#	, M#, MATTYP	)
ER273	( SUBSTR, LC#	, MATPAR	)
ER274	( SUBSTR, LC#	, ELTYP	)
ER275	( SUBSTR, LC#	, MATTYP	)
ER276	( SUBSTR, LC#	, CSTYPE	)
ER277	( SUBSTR, LC#	, DO#	)
ER278	( SUBSTR, LC#	, DESPAR	)
ER279	( SUBSTR, LC#	, D#	)
ER280	( SUBSTR, LC#	, DEVPAR	)
ER281	( SUBSTR, LC#	, EC#	)
ER282	( SUBSTR, LC#	, ECPAR	)
ER283	( SUBSTR, LC#	, SC#	)
ER284	( SUBSTR, LC#	, STCPAR	)
ER285	( SUBSTR, LC#	, MODE#	)
ER286	( SUBSTR, LC#	, MODPAR	)
ER287	( SUBSTR, LC#	, CTTYP	)
ER288	( SUBSTR, LC#	, ME#	)
ER289	( E#, LC#	, E#, ELTYP	)
ER290	( E#, LC#	, ELPAR	)
ER291	( E#, LC#	, M#, MATTYP	)
ER292	( E#, LC#	, MATPAR	)
ER293	( E#, LC#	, ELTYP	)
ER294	( E#, LC#	, MATTYP	)
ER295	( E#, LC#	, CSTYPE	)
ER296	( E#, LC#	, DO#	)
ER297	( E#, LC#	, DESPAR	)
ER298	( E#, LC#	, EC#	)
ER299	( E#, LC#	, ECPAR	)
ER300	( E#, LC#	, CTTYP	)
ER301	( QC#, LC#	, QC#	)
ER302	( QC#, LC#	, QLCPAR	)
ER303	( QC#, LC#	, EC#	)
ER304	( QC#, LC#	, ECPAR	)
ER305	( QC#, LC#	, NC#, VN	)
ER306	( QC#, LC#	, NDCVEC	)
ER307	( QC#, LC#	, SC#	)
ER308	( QC#, LC#	, STCPAR	)
ER309	( QC#, LC#	, CTTYP	)
ER310	( DOF#, LC#	, DOF#	)
ER311	( DOF#, LC#	, DOFPAR	)

Figure 7.3.5 (Continued)



relations together with original elementary relations are shown in the diagram of Fig. 7.3.6. Derived relations are shown in dotted lines in the figure.

### 7.3.3 Selecting Elementary Relations to Form a Conceptual Data Model

Now, we have uncovered all possible association between data used in the analysis and design computation. From the diagram of Fig. 7.3.6, we see that several paths for accessing a particular data are available. Out of these paths, suitable ones are selected according to our needs in the finite element analysis and optimization computations. The following paths are identified for some important data, and paths selected are discussed.

**Material Property Data.** Available paths are (a)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow M\# \rightarrow MATPAR$ , (b)  $S\# \rightarrow SUBSTR\# \rightarrow M\# \rightarrow MATPAR$ , and (c)  $S\# \rightarrow M\# \rightarrow MATPAR$ . Paths (b) and (c) are generally not useful unless all elements in a substructure or structure use the same material property. Path (a) identifies material property for each element and therefore it is selected.

**Cross-Section Data.** Available paths are (a)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow CSTYP \rightarrow CSDet$ , (b)  $S\# \rightarrow SUBSTR\# \rightarrow ELPAR$ , and (c)  $S\# \rightarrow ELPAR$ . Paths (b) and (c) are meaningless. Path (a) is appropriate to find cross-sectional property of a particular element.

**Degree of Freedom Data.** Available paths are (a)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow N\# \rightarrow DOF\#$ , (b)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow DOF\#$ , (c)  $S\# \rightarrow SUBSTR\# \rightarrow N\# \rightarrow$







DOF#, (d),  $S\# \rightarrow E\# \rightarrow DOF\#$ , and (e)  $S\# \rightarrow DOF\#$ . Path (a) is useful if nodal degrees of freedom are available. Path (b) is useful for assembly of stiffness matrix. Path (c) is useful for assembly of global matrices. Paths (d) and (f) are appropriate when substructures are not used. Paths (a), (b) and (c) are selected.

**Coordinate Data.** The coordinate data belongs to domain NODPAR and/or ELPAR. Available path to access NODPAR are (a)  $S\# \rightarrow SUBSTR\# \rightarrow N\# \rightarrow NODPAR$  (coordinates), (b)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow ELMVEC$  (coordinates), (c)  $S\# \rightarrow NODPAR \rightarrow E\# \rightarrow N\# \rightarrow NODPAR$ , (d)  $S\# \rightarrow ELPAR$ , and (e)  $S\# \rightarrow NODPAR$ . Paths (d) and (e) are meaningless. Path (a) just gives nodal coordinates of a substructure which may not be directly useful. Path (b) gives element coordinates which are useful for element stiffness computation. Path (c) is longer than path (b) to know coordinates of an element.

**Design Variable Data.** Available path to access data such as design value, upper bound, lower bound associated with  $D\#$  are (a)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow DG\# \rightarrow D\# \rightarrow DEVPAR$ , (b)  $S\# \rightarrow SUBSTR\# \rightarrow DG\# \rightarrow D\# \rightarrow DEVPAR$ , (c)  $S\# \rightarrow SUBSTR\# \rightarrow D\# \rightarrow DEVPAR$ , and (d)  $S\# \rightarrow D\# \rightarrow DEVPAR$ . Path (a) is useful to know the values of cross-sectional geometry of elements. Path (b) identifies the design values for each design group. Design variables of a substructure or structure are obtained by paths (c) and (d). All these paths are used in computation.

**Displacement Data.** To access displacements data (DOFPAR), the available paths are (a)  $S\# \rightarrow SUBSTR\# \rightarrow E\# \rightarrow N\# \rightarrow DOF\# \rightarrow DOFPAR$ , (b)  $S\#$



→ SUBSTR# → E# → DOF# → DOFPAR, (c) S# → SUBSTR# → N# → DOF# → DOFPAR, (d) S# → SUBSTR# → DOF# → DOFPAR, and (e) S# → DOF# → DOFPAR. Path (a) is longest to access displacement data. Path (b) is useful to know displacements of elements. Path (c) gives displacements of nodes. Paths (d) and (e) give displacement of a substructure or structure. These paths are used in computation and are retained.

Similar arguments are made in selecting various paths needed for computations. Each of the paths represents a set of elementary relations representing the conceptual data model for finite element analysis and structural design optimization.

#### 7.4 Design of Internal Data Model

An internal data model is designed for storing data of finite element analysis and structural design optimization data. The methodology developed in Chapter 4 is used to design this data model. The design of the internal data model is initiated by identifying the data needs of various computations listed in Section 2.3. The data so identified are arranged in a number of relations. These relations are refined by the normalization procedure. By this procedure, a new set of relations are obtained which are later checked for consistency with the conceptual (theoretical) data model. Note that this internal data model is used for implementing the database of structural design optimization program. Details of the internal data model design are given in Sections 7.4.1 and 7.4.2.



#### 7.4.1 Data Needed in Computation Process

The data needed and generated in various steps of analysis and design computation (see Section 2.3) are identified. They are arranged in a number of relations.

##### Element Level Computation.

R1 (E#, S#, SUBSTR#, N#, M#, ELTYP, MATTYP, CSTTYP, LC#, MN, VN, LDTYP, ELMAT, ELVEC, ELPAR, NODVEC, NODPAR, MATTAB, MATVEC, MATPAR)

##### Substructure Level Computation.

R2 (SUBSTR#, S#, E#, N#, DOF#, LC#, MODE#, MN, VN, LDTYP, SUBMAT, SUBVEC, SUBPAR, ELMAT, ELVEC, ELPAR, NODVEC, NODPAR, DOFVEC, DOFPAR, MODVEC, MODPAR)

##### Structure Level Computation.

R3 (S#, SUBSTR#, E#, N#, DOF#, LC#, MODE#, MN, VN, LDTYP, STRMAT, STRVEC, STRPAR, SUBMAT, SUBVEC, SUBPAR, ELMAT, ELVEC, ELPAR, NODVEC, NODPAR, DOFVEC, DOFPAR, MODVEC, MODPAR)

##### Recovery of Element Response.

R4 (E#, S#, SUBSTR#, N#, M#, ELTYP., MATTYP, CSTYP, LC#, MN, VN, LDTYP, STRVEC, STRPAR, SUBVEC, SUBPAR, ELMAT, ELVEC, ELPAR, NODVEC, NODPAR, MATTAB, MATVEC, DOFVEC, DOFPAR, MODVEC, MODPAR)

##### Design Problem Formulation.

R5 (D#, S#, SUBSTR#, E#, N#, M#, ELTYP, MATTYP, CSTYP, DG#, MN, VN, ME#, STRPAR, SUBPAR, ELVEC, ELPAR, NODVEC, NODPAR, MATTAB, MATVEC, DESVEC, DESPAR, DEVVEC, DEVPAR)

R6 (GC#, S#, SUBSTR#, E#, N#, M#, ELTYP, MATTYP, CSTYP, DOF#, LC#, DG#, D#, EC#, NC#, SC#, MODE#, MN, VN, CTTYPE, LDTYP, ME#, STRPAR, SUBPAR, ELVEC, ELPAR, NODVEC, NODPAR, DOFVEC, DOFPAR, DESVEC, DESPAR, DEVVEC, DEVPAR, GLCVEC, GLCPAR, ELCVEC, ELCPAR, NDCVEC, STCVEC, STCPAR, MODVEC, MODPAR)



### Constraint Checks.

R7 (GC#, S#, SUBSTR#, EC#, NC#, SC#, GLCPAR)

### Design Sensitivity Analysis.

R8 (GC#, S#, SUBSTR#, E#, N#, M#, ELTYP, MATTYP, CSTYP, DOF#, LC#, DG#, D#, EC#, NC#, SC#, MODE#, MN, VN, CTTYP, LDYP, ME#, STRMAT., STRVEC, STRPAR., SUBMAT, SUBVEC, SUBPAR., ELMAT, ELVEC, ELPAR, NODVEC, NODPAR, MATTAB, MATVEC, DOFVEC, DOFPAR, DESVEC, DESPAR, DEVVEC, DEVPAR, GLCMAT, GLCVEC, GLCPAR, ELCMAT, ELCVEC, ELCPAR, NDCVEC, STCVEC, STCPAR, MODVEC, MODPAR)

## 7.4.2 Relations and Matrices for Internal Data Model

The relations R1 to R8 formed in Section 7.4.1 are not in normal form. But they contain all the relevant data needed for the computation. Therefore, they are normalized using the methodology developed in Chapter 4. The normalized relations are shown in Fig. 7.4.2. These relations are consistent with the conceptual model designed in Section 7.3. The relations shown in Fig. 7.4.2 are grouped into several categories to identify them easily. The groups are (i) Element, (ii) Node, (iii) Material, (iv) Element Type, (v) Nodal Constraint, (vi) Element Constraint, (vii) Structure Constraint, (viii) Global Constraint, (ix) Mode, (x) Design Variable, (xi) Matrix, and (xii) Vector. Since, many of the relations are required independently for each substructure, substructure numbers are assigned to the relations to identify them.

The matrices used in the computation are organized using numerical data model. The internal model for numerical data is shown in Fig. 7.4.3. The model uses hypermatrix scheme to organize large matrices



ELDD-Y	ELEMENT LOAD VECTOR		Y - Substr#
Element Number (Name)	Load Case Number	Load Type	Vector Size
Ex			Pressure load

ELTP	ELEMENT TYPES	
Element Type (Name)	Element Characteristic Parameters	
Ex	No of Nodes/element; No DOF/Node Size of Stiff/Tranf Matrix; No. of Design Variables, C/S Shape; Gauss Pts No of data for C/S definition.	

ELRES-XY	ELEMENT RESPONSE VECTOR		X - Vector Name	Y - Substr#
Element Number	Load Case Number	Vector Size	Element Response Vector	
Ex			Displacement Vec, Velocity Acceln Vec, Stress Vec, Strain Vec, Force Vec, Instantaneous Elastic-plastic Stress-strain Matrix, Differential allowable Stress	

ELPAR-XY	ELEMENT PARAMETERS		X - Name	Y - Substr#
Element Number	Element Type	Element Parameters		
Ex		Material No.; Fixed Design Index Temp.; Design Group; Nonlin Index; Damage Flag; Element No. in each group.		

ELCON-XY	ELEMENT CONSTRAINTS		VECTORS	X-Vector Name	Y - Substr#
Element Number	Constraint; Vector	Type	Size	Element Constraint Vector	
Ex				stress Limit vector Element Numbers allowable stress	

ELVEC-XY	ELEMENT VECTORS		X - Vector Name	Y - Substr#
Element Number	Element Type	Vector Size	Element Vector	
Ex			Connecting Nodes, Design Variable Vector, C/S Geometry vector, Section Properties, Element DOF, Allow. St Curr Coord, Orien Vec	

MEMBR-Y	MEMBER DETAILS		Y - Substr#
Member Number	Design Group No	Element Number	Member Connectivity
Ex			Unbraced length effective length factor

ELMAT-XY	ELEMENT MATRICES		X - Matrix Name	Y - Substr#
Element Number	Element Type	Row Col	Element Matrix	
Ex			Stiffness/Transform; Mass Matrix, Damping Unit Stiff Coeff.	

Figure 7.4.2 Relations for Internal Model



NODPAR-Y NODAL PARAMETERS Y - Substr#			
Node Number	Nodal Parameters		
	Ex System Node Number, Global Node Number; Boundary node Number, General Boundary Condition; Index, Skew Coord Index, Temperature, Mid Surface vector Index , Special Coordinate number		
NODVEC-XY NODAL VECTORS X - Vector Name Y - Substr#			
Node Number or Index	Vector Size	Nodal Vector	
		Ex. Coordinate vector; DOF Vector; Mass Vector; Damping Vector; Boundary Condition Index; Mid Surface vector; Skew Coordinate Definition, Local Transformation for coordinates	
NODLOD-Y NODAL LOADS Y - Substr#			
Node Numbers	Load Case Number	Load Type Size	Nodal Load Vectors
NODRES-XY NODAL RESPONSE VECTOR X - Vector Name Y - Substr#			
Node Numbers	Load Case Number	Vector Size	Nodal Response Vectors
			Ex Nodal Displacement Vector Velocity Vector Acceleration Vector

NODCON-XY NODAL CONSTRAINTS X - Vector Name Y - Substr#			
Node Number	Constr Type	Vector size	Nodal Constraint Vectors
			Ex. Node Displacement Limits Node Number Vector Node velocity limits Node acceleration limits
MATPAR MATERIAL PROPERTY PARAMETERS			
Material Number	Material Type	Layer Numb	Material Property Parameters
			Ex. E, Mu, G, Density, Damping Coeff, Specific heat, Thermal coefficient; Cost Factor for Design; No. of layers; Type of Yield Criteria; Creep Law Type
MATVEC-XX MATERIAL PROPERTY VECTOR XX = Vector Name			
Material Number	Material Type	Vector Size	Material Property Vector
			Ex. Stress Limit Vector Fibre direction vector Layer Thickness vector Creep Constant Vector Yield stress Vector
MATMAT-XX MATERIAL PROPERTY MATRIX XX= Vector name			
Material Number	Material Type	Temperature Size	Row Col Material Property Matrix
			Ex. Youngs Modulus Table; Poissons Tab Therm. Coeff Table Strain Hard Table

Figure 7.4.2 (Continued)



```

=====
SUBMAT-X : STRUCTURE PARAMETERS X - Integer/Real Index
=====
Substructure Number
=====
Ex Number of Substructures
Number of load cases
Number of Materials used
=====

```

```

=====
STRVEC-X : STRUCTURE VECTORS X - Vector Name
=====
Load Case Number: Vector Size : Structural Vectors
=====
Ex Load Vectors
Displacement Vectors
Acceleration Vector
Velocity vector
=====

```

```

=====
STRMAT-X : STRUCTURE MATRIX X- Matrix Name
=====
Hyper Row: Hyper Column: Structure Matrix
Number Number
=====
Ex Stiffness Matrix
Mass Matrix
Load Matrix
Damping Matrix
=====

```

```

=====
STANDON-XY : NONLINEAR STRUCTURE MATRIX X - Name
AND DESIGN SENSITIVITY MATRIX
=====
Load Case:Hyper Row:Hyper Column : Substructure Submatrices
Number Number Number
=====
Ex Tangential Stiffness
Matrix, Geometric Stiffness
Matrix, d z/db, lamda
=====

```

```

=====
SUBMAT-XY : SUBSTRUCTURE PARAMETERS X - Integer/Real Index
=====
Substructure Number
=====
Ex Number of elements in substructure
Number of nodes in substructure
Equivalent substructure number
Fixed Design Index
=====

```

```

=====
SUBVEC-XY : SUBSTRUCTURE VECTORS X - Vector Name Y - Substr#
=====
Load Case Number: Vector Size : Substructure Vector
=====
Ex Load Vector, Effec Load Vector
Internal, Boundary Displac
Acceleratn, Velocity Vector
dF/db, Effective Force Grad
=====

```

```

=====
SUBMAT-XY : SUBSTRUCTURE MATRIX X- Matrix Name Y - Substr#
=====
Hyper Row: Hyper Column: Substructure Submatrices
Number Number
=====
Ex Internal Stiffness Matrix
Boundary Stiffness Matrix
Internal-Boundary Stiffness Matrix
Condensed Stiffness Matrix
Mass Matrix, Load Matrix
=====

```

```

=====
SUBNON-XY : NONLINEAR SUBSTRUCTURE MATRIX X - Name Y - Substr#
AND DESIGN SENSITIVITY MATRIX
=====
Load Case:Hyper Row:Hyper Column : Substructure Submatrices
Number Number Number
=====
Ex Tangential Stiffness Matrix
Geometric Stiffness matrix,
d h/db, d z/db, lamda 1
Cl matrix, C2 matrix
=====

```

Figure 7.4.2 (Continued)



NDCONS-Y NODAL CONSTRAINTS VALUES				Y - Substr#			
Nodal Number	Constraint Number	Node Number	Dof: Constr Type	Constr Value	Active Code	DOF Constr Gradients	

FRCON - Y FREQUENCY CONSTRAINT VALUE				Y - Substr#			
Frequency Constraint No.	Mode No.	Actv Code	Constr Value	Constr Value	Constr Vector	Constr Gradient	

SENCF-XY : SENSITIVITY COEFFICIENTS				X - Vector Name Y - Substr#			
Global Constraint Number	Vector Size	Sensitivity Coefficients					

COSTGRD-Y COST GRADIENTS				Y - Substr#			
Design variable Number						Cost Gradients	

DESBND-Y DESIGN VARIABLE BOUNDS				Y - Substr#			
Design Variable Number	Design Fix/Free Index					Design Bound Vector	

DESNUM DESIGN VARIABLE NUMBERING							
Global Design Variable No	Substructure Number					Local design variable No	

ELCONS-Y ELEMENT CONSTRAINTS				Y - Substr#			
Element Stress Constraint	Element Index	Constraint Type				Active Index	Element Constr Gradnt







100

[illegible]

THE UNIVERSITY OF CHICAGO

100

204



assembled at substructure and structure level. Skyline and banded scheme for large matrix data organization are also shown in Fig. 7.4.3. Various structure level and substructure level matrices are shown in Fig. 7.4.2.

### 7.5 An External Data Model Design

Building a complete set of relations forming an external data model is entirely dependent on the requirements of new applications that will use the database. Considering all possible new applications and their requirements are beyond the scope of this study. Hence, no further attempt is made here to derive all possible relations for an external data model. However, some specific examples are given below to describe external model design.

Suppose, a new application needs the data of element degrees of freedom numbers. Assuming that this data is not already stored in the database, a new relation is constructed containing the required data. In this case, a new relation ELMDOF is derived as shown in Fig. 7.5.1 from existing relations ELMVEC and NODVEC. Similarly, if another application needs the element connectivity data separately for each type of element, then relations BEAM, TRIANG, and QUADRAL are derived as shown in Fig. 7.5.1.

### 7.6 Evaluation of Database Design Methodology

Database design methodology developed in Chapter 4, is found to be extremely useful in designing the database given in Sections 7.2 to



# External Model for Element Nodal Connectivity

```

To obtain      Do
-----
ELMDOF         INTERSECT ELMVEC WITH NDDVEC FORMING ELMDOF USING
                E# N# DOF1 DOF2 ... DOFN

ELMCMOR        INTERSECT ELMVEC WITH NDDVEC FORMING ELMDOF USING
                E# N# COORD-VEC

ELMDAT         JOIN ELMCMOR USING E# WITH ELMPRO USING E# FORMING
                ELMDAT WHERE EQ

```

### External Model for Element Related Data.

### Figure 7.5.1 Relations for External Model



7.5. The methodology is used to design the conceptual data model for finite element analysis and structural optimization data. It uses information about entities, domains, functional dependencies in data to obtain the data model. The conceptual data model enabled us to understand the inherent nature of finite element analysis and structural design optimization data. Thus, we are able to associate different data items depending on their characteristics. Many associations of data items which cannot be readily identified by the database designer was brought out by the use of transitive closure principle and connectivity matrix procedure. Thus, the methodology to develop the conceptual data model for finite element analysis and design optimization data is found useful. This conceptual data model serves as a theoretical database to check the database that is designed using the internal data model.

However, in using the methodology to design the conceptual data model, some difficulties are encountered. One of the main difficulties is in the proper identification of entities and domains of finite element analysis and design optimization data. For example, data of element type -- is it a separate entity or just a property of entity element? Similarly, data of cross-section type faces the same problem. The methodology of database design did not suggest suitable schemes to classify such kind of data. Another problem was to what extent the generality of domain identification should be maintained. At one extreme, domains may be identified as integer numbers, real numbers, characters, vectors and matrices. At another extreme, domains may be identified in great detail, for example: a set of element



numbers greater than 1 and less than 100, a set of stiffness matrices of size  $n \times n$ ; a set of load vectors of size  $n$ ; a set of truss element member connectivity vector. Too much generality will not bring out clear identification of characteristics of finite element analysis and design optimization data. Detailed approach in identifying domains will lead to confusion and tedious database design process. Further, in selecting a suitable set of elementary relations from the transitive closure, some difficulties are encountered. Since, there exists several path to access data, selection of a suitable path depends on the judgement of database designer. Use of processing sequence has helped in resolving selection of elementary relations to some extent.

The methodology to design an internal data model for finite element analysis and structural optimization data has provided us with a tool to design a database and implement it using a DBMS. The methodology is found very useful in separating general tabular type of data and matrix data used in analysis and optimization computations. The use of normalization procedures in internal database design has enabled us to obtain a set of relations which were consistent with theoretical model and thereby eliminating redundancy in data storage.

One of the problems encountered in using the methodology to design relations for internal model is in deciding whether all attributes associated with a particular key attribute be combined in one relation or separated into two or more relations. For example, in the relation ELPAR, is it useful to include attributes: Nonlinear index and damage flag, even when nonlinear analysis or damage members are not used in



computation. If included, they introduce redundancy in data storage, but help in using the same relation for other types of computations. Another problem is faced in deciding how many relations to use for storing data; for example, connecting nodes of an element. If connecting nodes of all types of elements are stored in one relation, then we need to use variable length vectors which requires more overhead information to access data. On the other hand, if separate relations are used for each type of finite element, then number of accesses to different relations increases introducing inefficiency in data access.

The methodology suggests schemes to satisfy data requirements of different applications by using an external data model. Even when the implemented database does not have the data required by certain applications in the form they need, the methodology suggests adding new relations to provide the required data. But by adding, new relations containing data derived from existing relations introduces redundancy in data storage. The methodology does not suggest solution to this problem.

In summary, the database design methodology developed for finite element analysis and structural design optimization is found to be extremely useful. Intuitive methods of database design can be replaced by a systematic approach for the most part.



## CHAPTER 8

### IMPLEMENTATION OF A COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION SYSTEM USING DBMS

#### 8.1 Introductory Remarks

A computer-aided structural design optimization system is implemented using the database designed in Chapter 7 and the database management system MIDAS. This implementation is carried out to demonstrate use of the database and the database management system in finite element analysis and structural design optimization applications. Also, the results of the implementation will be used in evaluating DDL, DML, query language, database design, performance of DBMS in equation solving environment, and performance of design optimization program.

This chapter describes details of the computer program for structural design optimization. The program is developed in two parts. The first part is a general purpose finite element analysis program. The second part is a design sensitivity analysis program. In Section 8.2, the capabilities of the program are described. The program design is discussed in Section 8.3. Finally, the example problems solved using the program are given in Section 8.4. Evaluation of DBMS is discussed in the next chapter.



## 8.2 Capabilities of the Program

The general purpose finite element analysis and design sensitivity programs have the following capabilities:

1. The program can perform static analysis, design sensitivity analysis and optimal design of a structure.
2. The program can compute displacements, stresses, cost and constraint function values, cost and constraint gradients, impose stress, displacement and design variable constraints on a structure.
3. The program has an element library consisting of two and three dimensional truss, beam, triangular membrane, quadrilateral membrane and shear panel elements. New elements can be easily added into the library.
4. Hypermatrix scheme is used for assembly of large matrices in finite element analysis and design sensitivity computations.
5. Out-of-core solution of equations is carried out using hypermatrix scheme.
6. The program has modular structure. Modules in the program are functionally independent. Modification of existing modules and addition of new modules are simple to carry out.
7. The program uses a well designed database and the database management system MIDAS.
8. Any large size structure can be analyzed and designed using the program.



9. Program uses IDESIGN3 (Arora, et. al., 1984b) for finding the design change and optimal design computation. The program IDESIGN3 has options to select mathematical programming methods, like Hybrid, RQP and cost-function bounding.

### **8.3 System Design**

In this section, details of computer program design for finite element analysis and structural design optimization are given. Function and description of modules used in the program are given.

#### **8.3.1 DBMS Used in the Program**

Data Management System - MIDAS/R is used in implementing the finite element and design sensitivity programs. The subroutines listed in Section 6.2 are used for database management operations. The use of MIDAS/R enables us to evaluate its performance in design optimization applications. In a parallel research study, the performance of MIDAS/N for finite element analysis and optimal design is being evaluated.

#### **8.3.2 Finite Element Analysis Program**

The general purpose finite element analysis program developed has several modules. They are (i) input module (INPUT), (ii) data generation module (STRMGN), (iii) module for element stiffness computation (ELSTF), (iv) module for assembly of stiffness matrix (STIFF), (v) module for assembly of load matrix (LOAD), (vi) module for solution of equations (SOLEQ), and (vii) module for recovery of



displacements and stresses (STRESS). These modules are functionally independent. They directly interact with the database for input and output of data using DBMS. Various modules used in the program are schematically shown in Fig. 8.3.1. The function and description of the modules are given in the following paragraphs.

**Input Module.** The input module reads the data used in the finite element idealization such as material property, element connectivity, nodal coordinates, boundary conditions, and cross-sectional property. The data read is stored in the database for further processing. Data for each substructure is given separately. The module has subroutines INTIAL, DATAIN, DATELT, DCORNS, DATEPR, DATCON, ESRELT, SETDF1, SETDF2, and SEIDF3. Subroutine INTIAL, reads the general data needed for a structure such as material properties, element types, and number of substructures. DATAIN reads general data of a substructure such as number of elements, and number of nodes. Element connectivity data and cross-sectional properties are read in DATELT. Coordinate data and boundary conditions are read in subroutine DCORNS. Subroutines DATEPR, DATCON and ESRELT store the input data in various relations. Subroutines SETDF1, SETDF2, and SETDF3 define relations in a database. These subroutines are again functionally independent. They call MIDAS/R routines for database management operations.



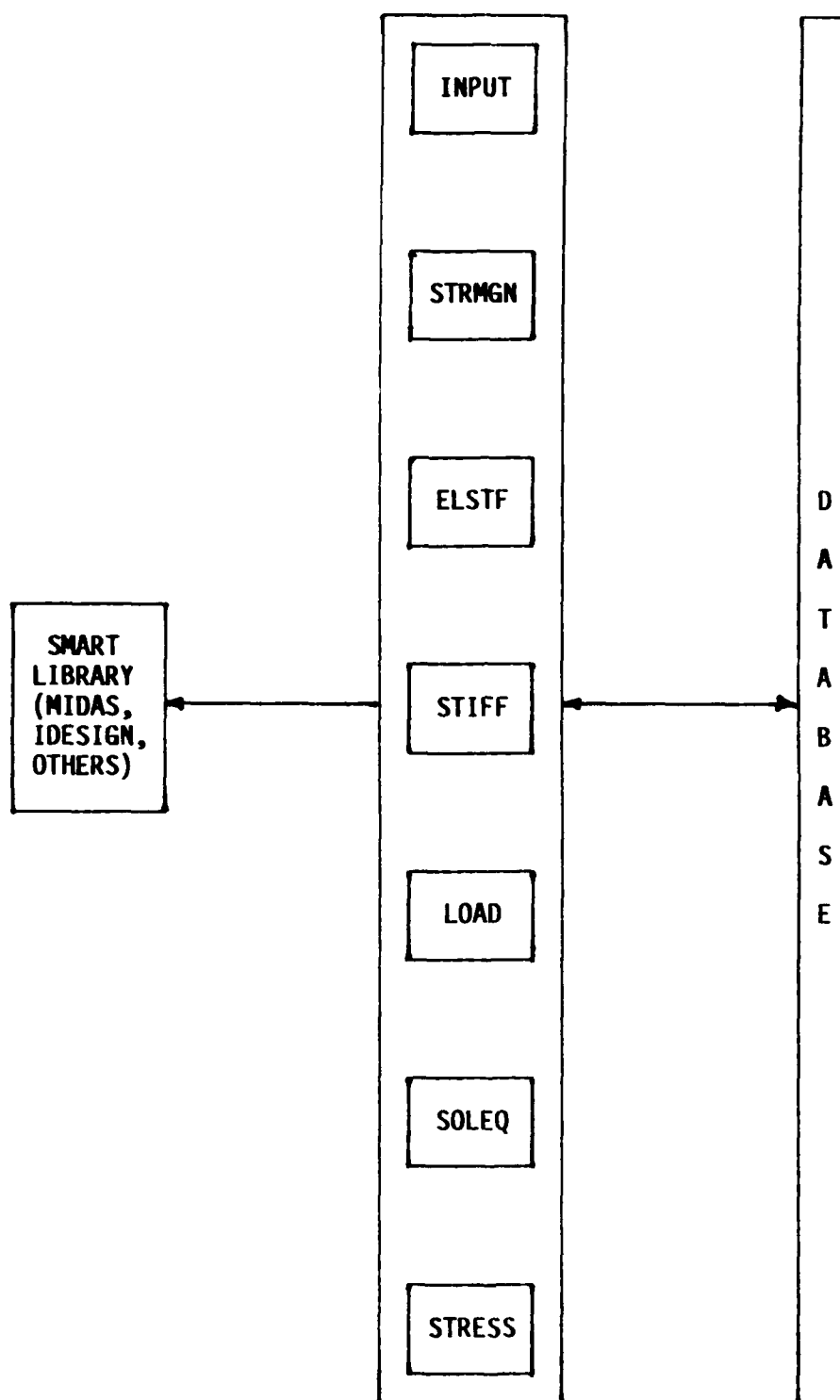


Figure 8.3.1 Modules in Finite Element Analysis Program



**Module for Data Processing.** This module generates data such as degrees of freedom numbers for nodes, degrees-of-freedom numbers for elements, element number participating in submatrices of assembled stiffness matrix, address of submatrices of the hypermatrix (stiffness) and degrees-of-freedom numbers for internal and boundary nodes of a substructure. The module has subroutines INBDDF, PRSCDF, ELTBOL, ELTPGS, FMNEIP, LAHMYT, ENISMP, SETDF4, SETDF5 and SETDF6. The module uses the relations created from the previous module to generate data and stores them in new relations. Subroutine INBDDF and PRSCDF generate degrees-of-freedom numbers at various nodes of a substructure. Subroutine ELTBOL generates degrees-of-freedom numbers of elements. Subroutines ELTPGS, LAHMYT and ENISMP are used for generating information about nonnull partition numbers in assembled stiffness matrix, element numbers contributing stiffness to submatrices of the hypermatrix etc. Subroutines SETDF4, SETDF5 and SETDF6 are used for defining relations in the database.

**Module for Element Stiffness Computation.** Element stiffness and transformation matrices are computed in this module. The module has a library of subroutines to generate stiffness matrices for various finite elements used in the program. The library contains routines for two and three dimensional finite elements - truss, beam, triangular membrane, quadrilateral membrane, and shear panel elements. The subroutines used in the module are ETRUSS, EBEAM2, EBEAM3, ETRMEM, EQDMEM, SMSPAR, ESSP, TRANS1, TRANS2 and TRANS3. ETRUSS generates element stiffness matrix for two and three dimensional truss elements in local and global



coordinate system. Element stiffness matrix for two and three dimensional beam elements are generated in subroutines EBEAM2 and EBEAM3. Element stiffness matrix for triangular membrane, quadrilateral membrane, spar and shear panel elements are generated in ETRMEM, EQDMEM, SMSPAR, and ESSP respectively. Transformation matrices required for these elements are computed in TRANS1, TRANS2 and TRANS3. Options are available either to store the element stiffness matrix or to generate it as and when required. New elements can be added easily into the library.

**Module for Assembly of Stiffness Matrix.** This module assembles stiffness matrix of a structure. The module has capability to assemble any type of finite element used in a structure. The program assembles the stiffness matrix in a hypermatrix form. The submatrices are numbered as shown in Fig. 8.3.2. One submatrix is assembled at a time using information about element stiffness matrices contributing to the submatrix. This scheme, therefore, reduces the number of I/O required to assemble a matrix. The program has capability to assemble internal and boundary stiffness matrices required for substructure analysis. The module has subroutines ASMBLY and SETDF11. ASMBLY subroutine actually assembles the stiffness matrix for a substructure in the hypermatrix form. Options are provided in the module either to use the element stiffness matrix data from the database or compute them when required. Null submatrices are not assembled.



INTERNAL DOF						BOUNDARY DOF	
1	2	4	7	11	16	22	28
	3	5	8	12	17	23	29
		6	9	13	18	24	30
			10	14	19	25	31
				15	20	26	32
					21	27	33
						34	35
							36

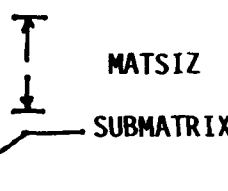

  
 MATSIZ  
 SUBMATRIX

Figure 8.3.2 Assembled Stiffness Hypermatrix



**Module for Assembly of Load Matrix.** Load matrix is assembled in this module. Any number of load cases can be assembled using the module. Again the assembly of load matrix is in the hypermatrix form. This module reads the load data and stores them in relations. The load data is processed according to degrees-of-freedom and load case numbers, and stored in relations. The load assembly routines assembles one load submatrix at a time using the data created by an earlier routine. The module has capability to assemble internal and boundary load matrices of a substructure separately. The subroutines in the module are LOAD1, LOAD2, LOAD4, LOAD5, SETDF7, and SETDF8. Subroutine LOAD1 reads the loads acting at various nodes of a substructure for different load cases. Subroutines LOAD2 and LOAD4 process the load submatrix numbers and other intermediate data required for load hypermatrix assembly. Load matrices are assembled for each substructure in subroutine LOAD5. SETDF11 routine defines relations needed for LOAD module. The load submatrix numbering is schematically shown in Fig. 8.3.3. Null submatrices are not assembled.

**Module for Solution of Equations.** Large system of simultaneous equations are solved in this module. The module uses the stiffness and load matrices assembled in the hypermatrix form. Out-of-core solution of equations is used. The program decomposes stiffness matrix and performs forward and backward substitutions on the assembled load matrix. Subroutines used in the module are DRIVE, FRMVEC, FRMVEX, MODIFY, GETHYP, PUTHYP, TRIPLE and MLTPY. Subroutines, DRIVE and MODIFY decompose stiffness matrix.



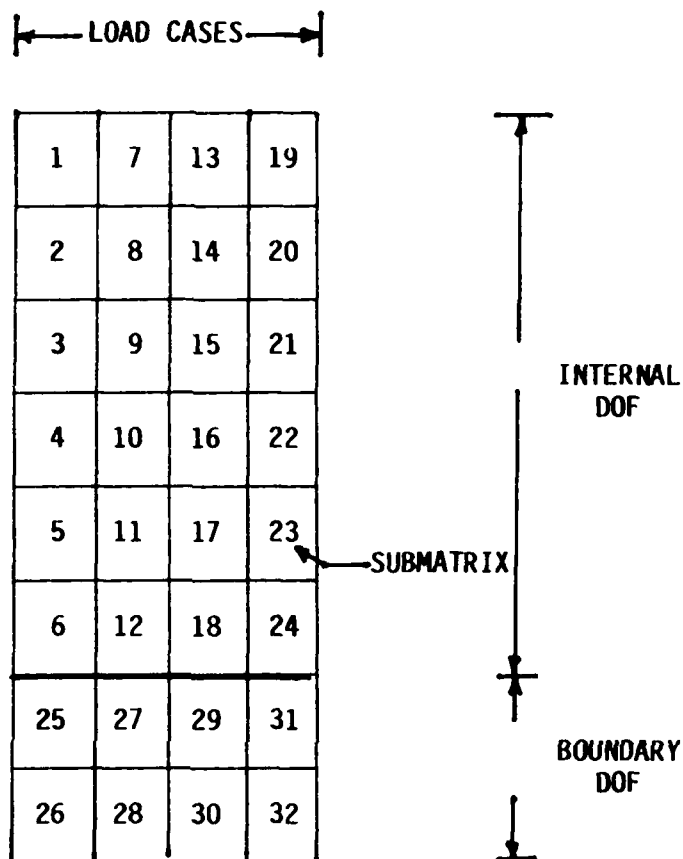


Figure 8.3.3 Load Hypermatrix



Subroutines FRMVEC and FRMVEX perform forward and backward substitutions on the load matrix. PUTHYP and GETHYP store and retrieve submatrices of assembled matrix. Subroutines MLTPLY and TRIPLE are used for multiplication and triple multiplication of submatrices.

**Module for Recovery of Displacements and Stresses.** This module recovers the displacement results from the solution module and rearranges them node-wise and load case-wise. Again the nodal displacements are rearranged to get displacements at the element level. The module also computes stresses in the elements using the element-displacement information. The subroutines used in the module are DISPI, DISPE, ETRMEM and ETRUSS. Nodal and element displacements are recovered in subroutines DISPI and DISPE respectively. Stresses for triangular membrane and truss elements are computed in ETRMEM and ETRUSS respectively.

### 8.3.3 Design Sensitivity Analysis Program

Design sensitivity analysis program has several modules. They are (i) cost function module (COST), (ii) constraint function module (CONFUN), (iii) cost gradient module (CSTGRD), (iv) partial derivatives of element and node related constraints (PARDV), (v) assembly of  $dh/db$  matrix (DHDB), (vi) solution for  $dz/db$  (DZDB), and (vii) constraint gradient module (CONGRD). These modules are functionally independent. They directly interact with the database for accessing data required for computation. The various modules used in the program are shown in



Fig. 8.3.4. The function and description of these modules are given in the following paragraphs.

**Cost Function Module.** This module computes the cost of a structure. The module computes the volume of each element used in the structure and multiplies by its material density to obtain the mass of the structure. Subroutine CSTTRS is used for computing mass of each truss elements. Module has provision to add cost function for other types of elements.

**Constraint Function Module.** This module computes constraint function of element related and node related constraints. The module generates constraint numbers corresponding to element and node related constraints. Also, global constraint numbers are generated. Element related constraints are stress limits on elements. Node related constraint functions are displacement limit at various degrees of freedom. The module has subroutines CONFUN, ELMSTR, and ETRUSS. Subroutine CONFUN generates constraint numbers and stores them in the database. Subroutine ETRUSS computes stress constraint for each truss element. Stress constraints for other types of elements can be added into this module.

**Cost Gradient Module.** Cost gradient of a structure is computed in this module. The cost gradient computation are carried out with respect to each design variable and stored in a vector. Cost gradients for truss elements are computed in subroutine CSTTRS.



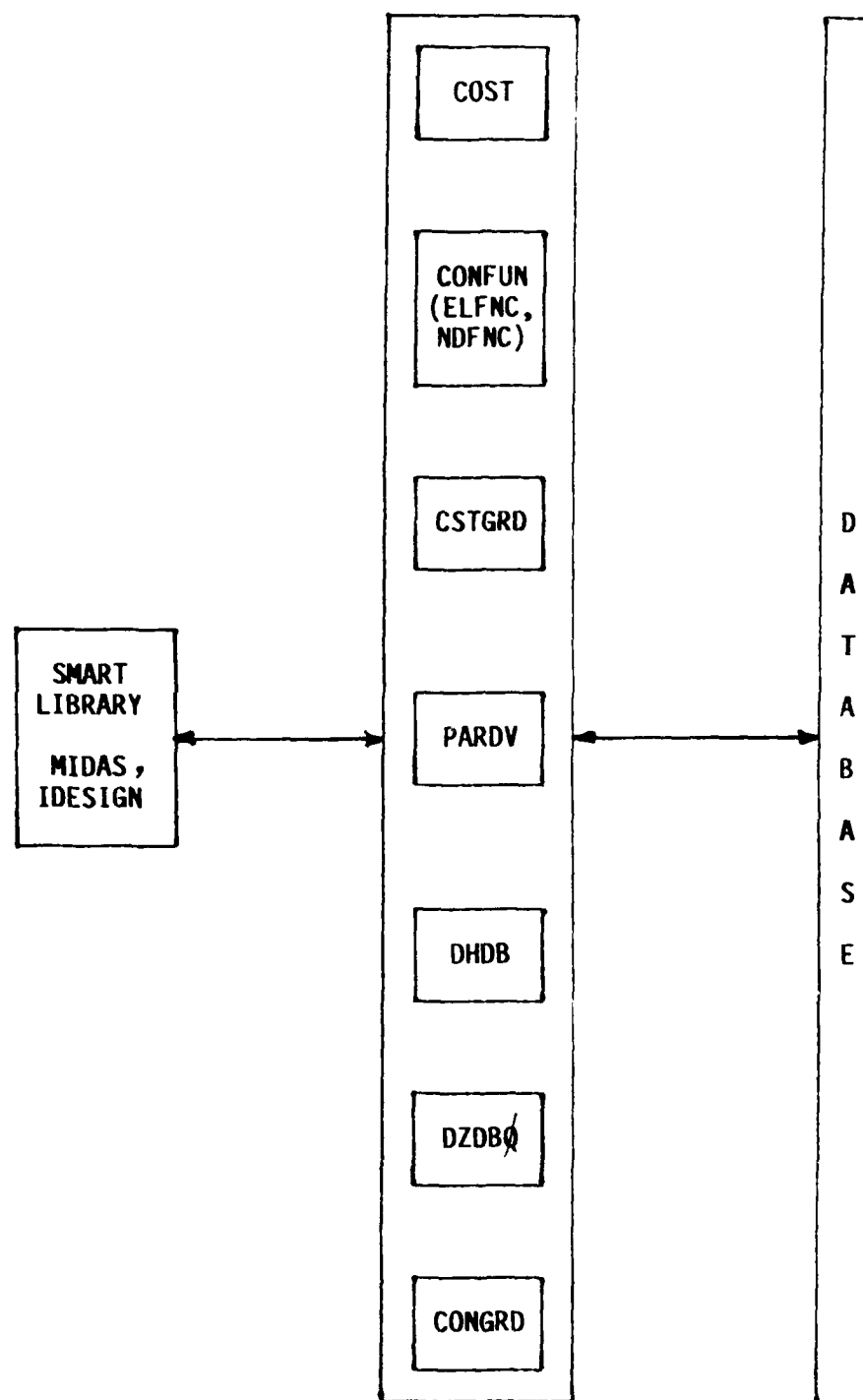


Figure 8.3.4 Modules in Design Sensitivity Analysis Program



**Partial Derivatives of Element and Node Related Constraints.** The partial derivatives of element and node related constraints with respect to displacements are computed in this module. The module has subroutines ELMSTR and ETRUSS to compute derivatives of constraints for truss element. The derivatives are stored in the database.

**Module DHDB.** The module assembles  $dh/db$  matrix for a structure. The module computes unit stiffness matrix of various elements in the structure and multiplies with corresponding displacements and assembles the matrix. The matrix is assembled in the hypermatrix form. The module has subroutines ASMDHB, STFGEN, and ETRUSS. Subroutine ASMDHB assembles the  $dh/db$  in hypermatrix form. Assembly procedure is similar to that for the stiffness matrix. Here the rows of the assembled matrix correspond to degrees of freedom numbers and the columns correspond to design variable numbers. Subroutine STFGEN calls ETRUSS or other routines depending on the element type. Subroutine ETRUSS computes unit stiffness matrix for truss elements and multiplies with the corresponding element displacements and returns the matrix  $\frac{dk}{db} \cdot z$ .

**Module DZDB.** This module obtains solution for  $dz/db$  using decomposed stiffness matrix and  $dh/db$  matrix. The solution procedure is the same as described for solution of displacements. Out-of-core solution of equations is carried out using the assembled hypermatrices. The module has subroutines FRMVEC, FRMVEX, MLTPLY, TRIPLE, PUTHYP and GETHYP. Subroutine FRMVEC and FRMVEX perform forward and backward substitutions on right hand side matrix. Subroutine MLTPLY



and TRIPLE perform multiplication and triple multiplication of submatrices. PUTHYP and GETHYP stores and retrieves submatrices in the database.

**Constraint Gradient Module.** Constraint gradients of a structure are computed in this module. The module multiplies the vectors  $\partial\psi/\partial\mathbf{z}$  and  $d\mathbf{z}/d\mathbf{b}$  to obtain constraint gradients  $d\psi/d\mathbf{b}$  of a structure. The constraint gradients are stored in the database for use in IDESIGN3 program. These computations are done in subroutine SENSTV.

**IDESIGN3 Program.** IDESIGN3 program (Arora and et. al., 1984b) is used to solve the minimization problem. The cost and constraint functions and their gradients are supplied to the program through the database. The design change values returned by the program are stored in the database. With the new values of the design variables, the finite element analysis and design sensitivity computations are repeated. CPL commands of PRIME computer system are used to run the program iteratively.

#### 8.4 Example Problems Solved Using the Program

Several example problems are solved using the structural design optimization program. These examples enable us to evaluate several aspects of the computer-aided structural design optimization process. First of all, the examples show that many practical structural optimization problems can be solved using the program. Secondly, the performance of the program can be evaluated. Thirdly, it is possible to



evaluate the use of database and database management system in the program.

Problems solved using the program are (i) 10-bar truss (ii) 25-bar truss (iii) 47-bar truss (iv) 72-bar truss (v) 200-bar truss (vi) 108-bar helicopter tail boom. The problems solved are shown in Figs. 8.4.1 to 8.4.6. Data for these problems were obtained from Thanedar, Park and Arora (1985). The optimum cost results obtained using the program was found to agree with those given in the reference. The intention of solving these problems was to evaluate the use of database, database management system and the computer program efficiency. Results of evaluation are given in Chapter 9.



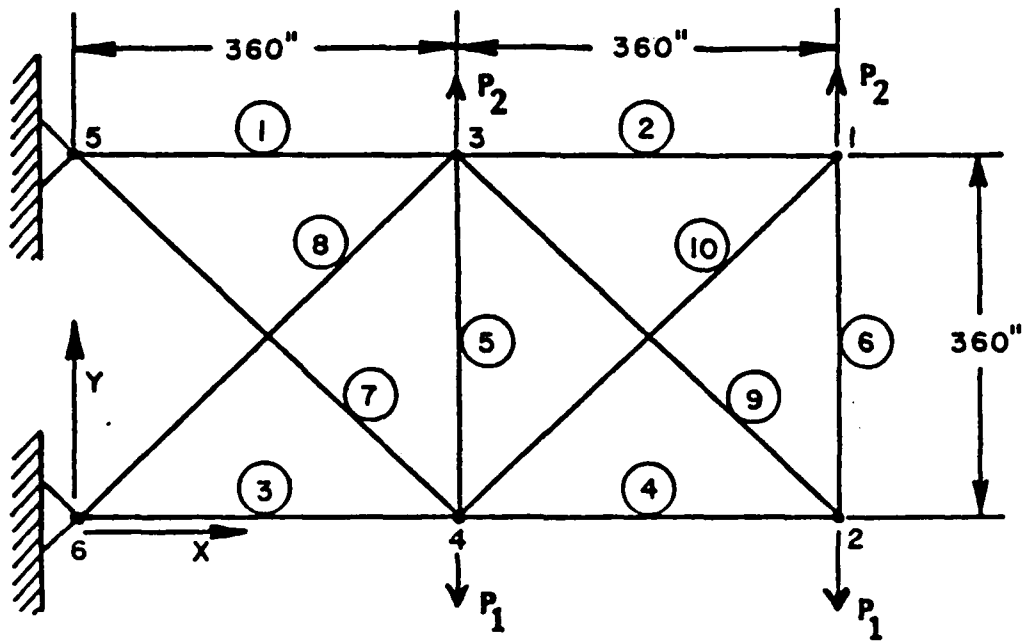


Figure 8.4.1 Ten-Bar Truss



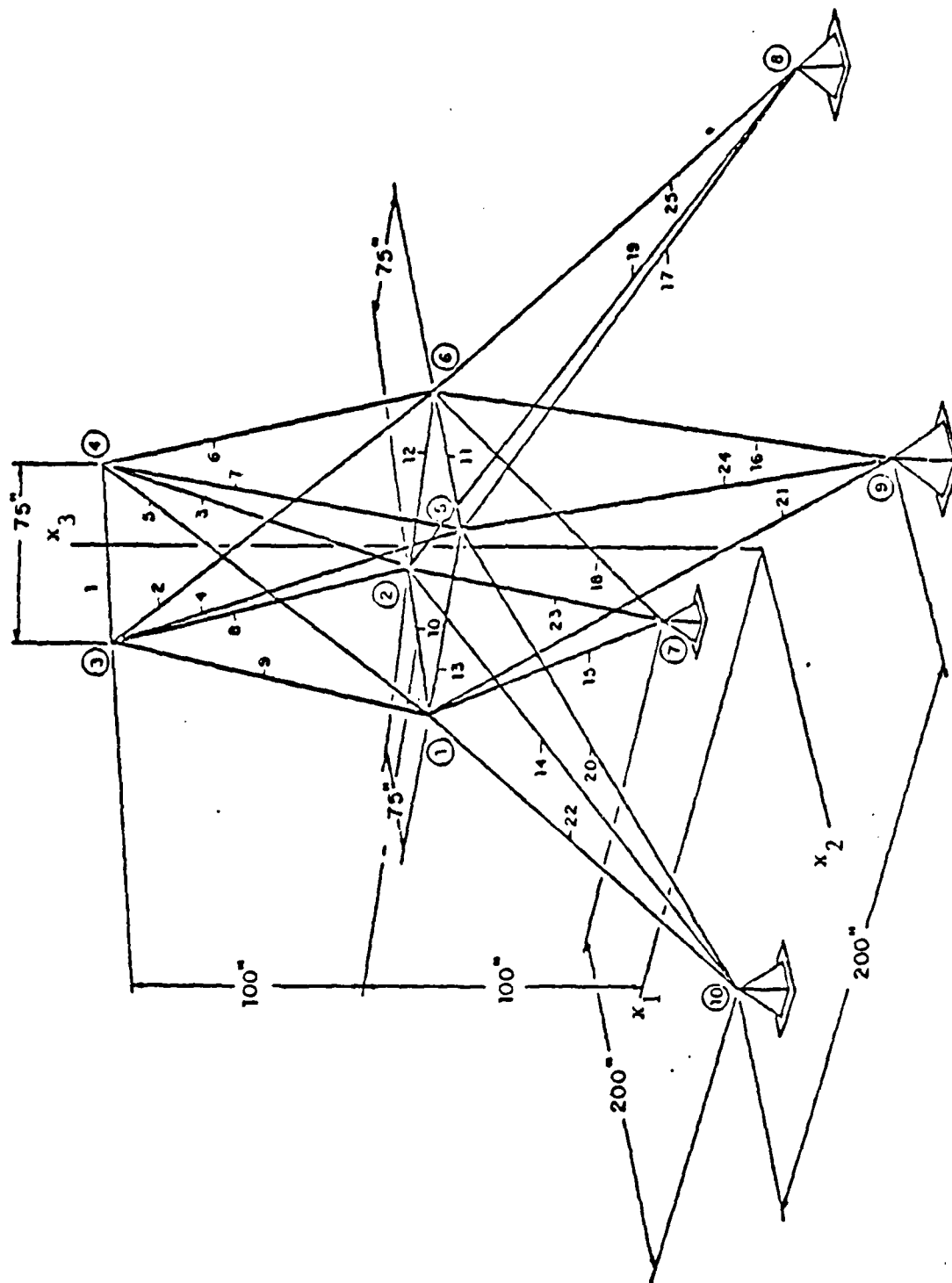


Figure 8.4.2 Twenty-five Member Transmission Tower



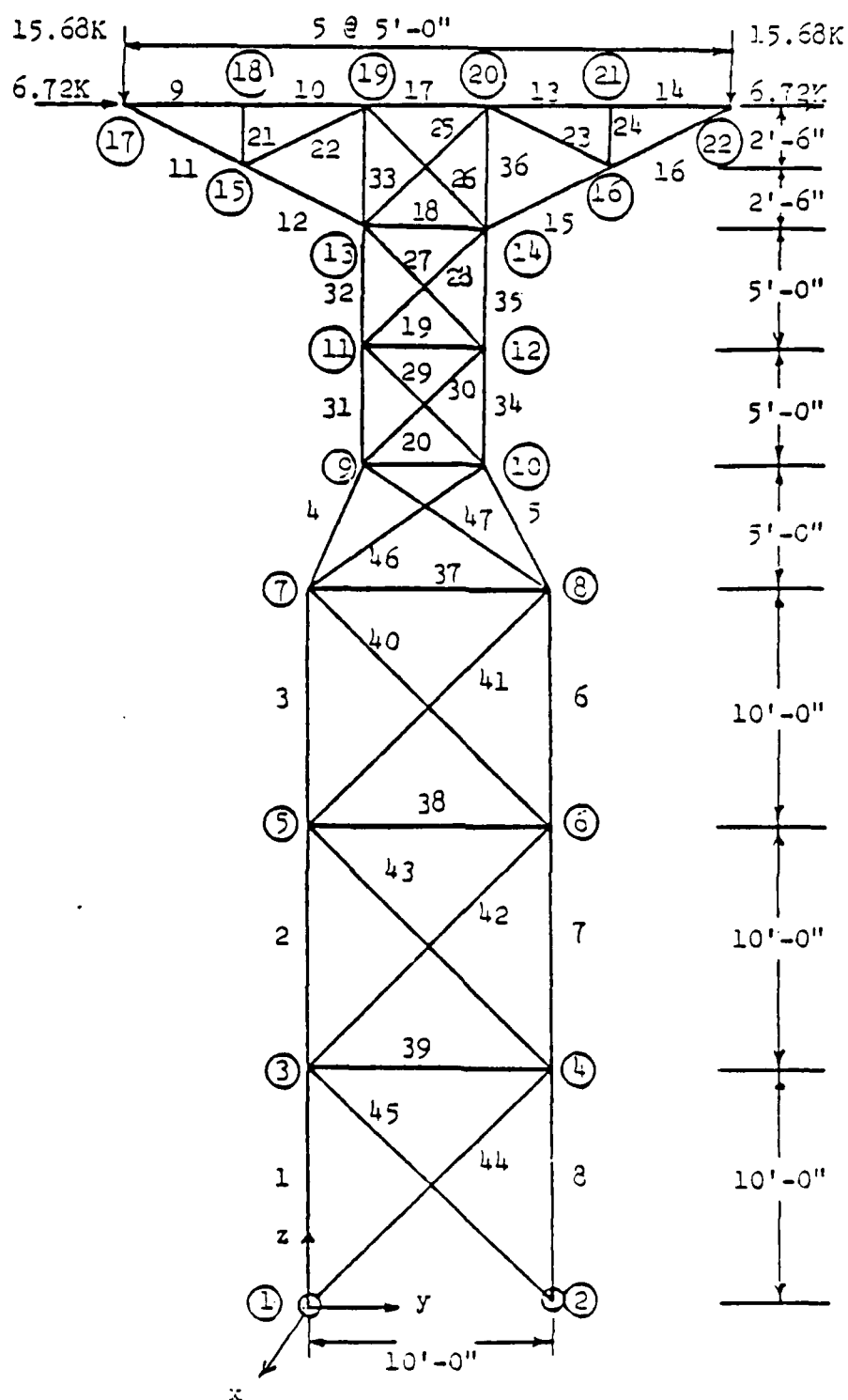


Figure 8.4.3 Forty-seven-Member Plane Truss



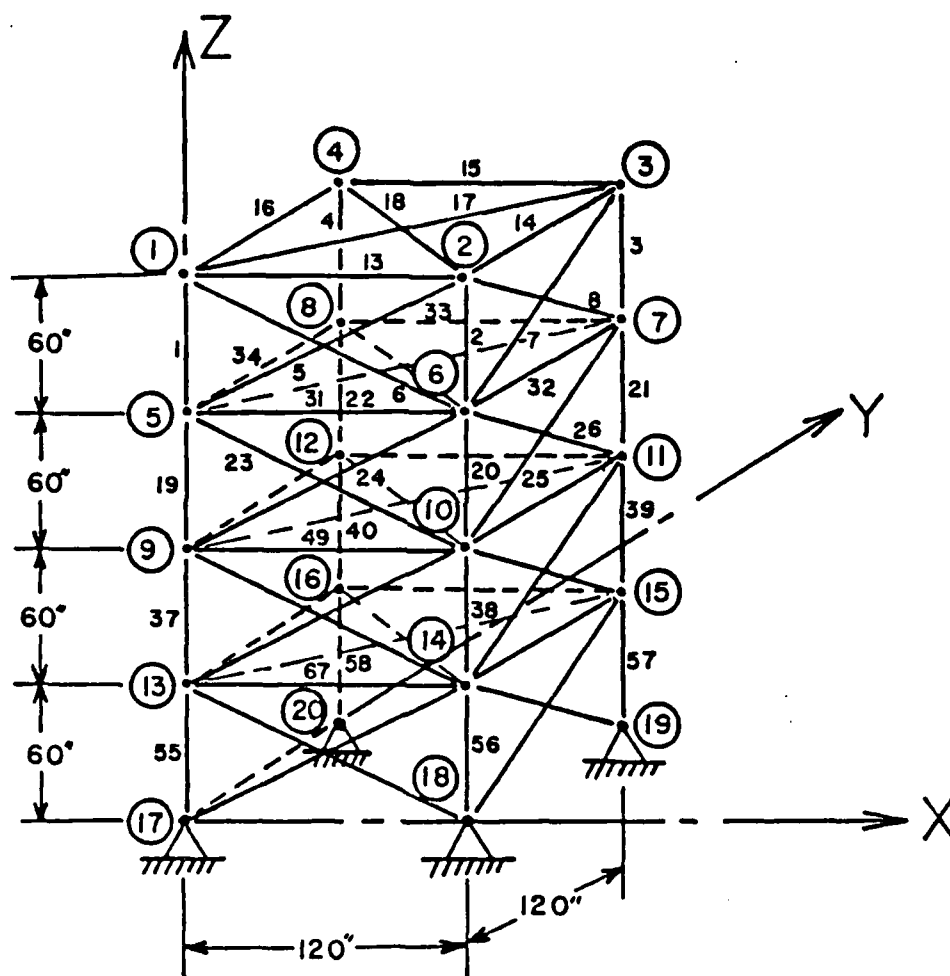


Figure 8.4.4 Seventy-two-Member Space Truss



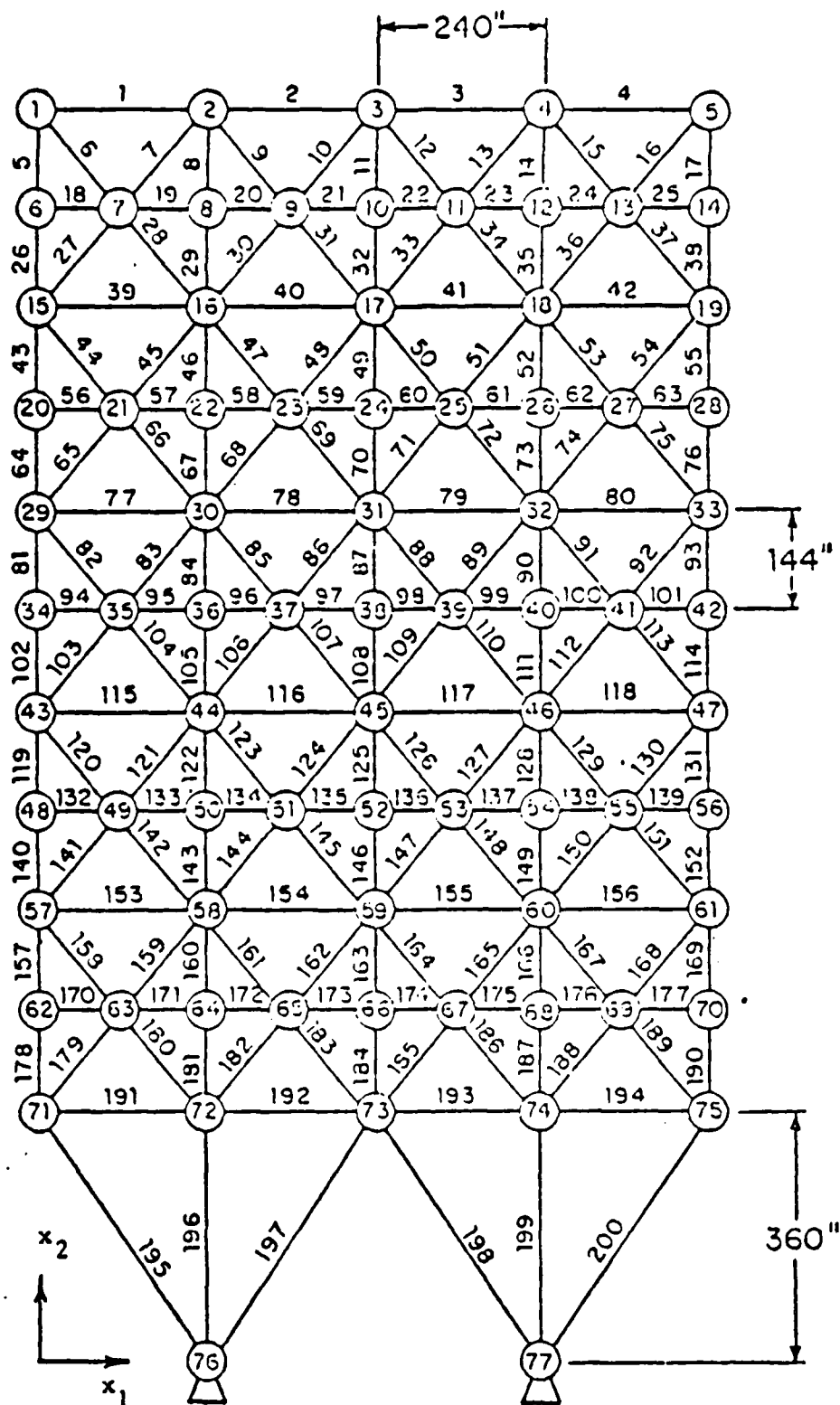


Figure 8.4.5 Two Hundred-Member Plane Truss



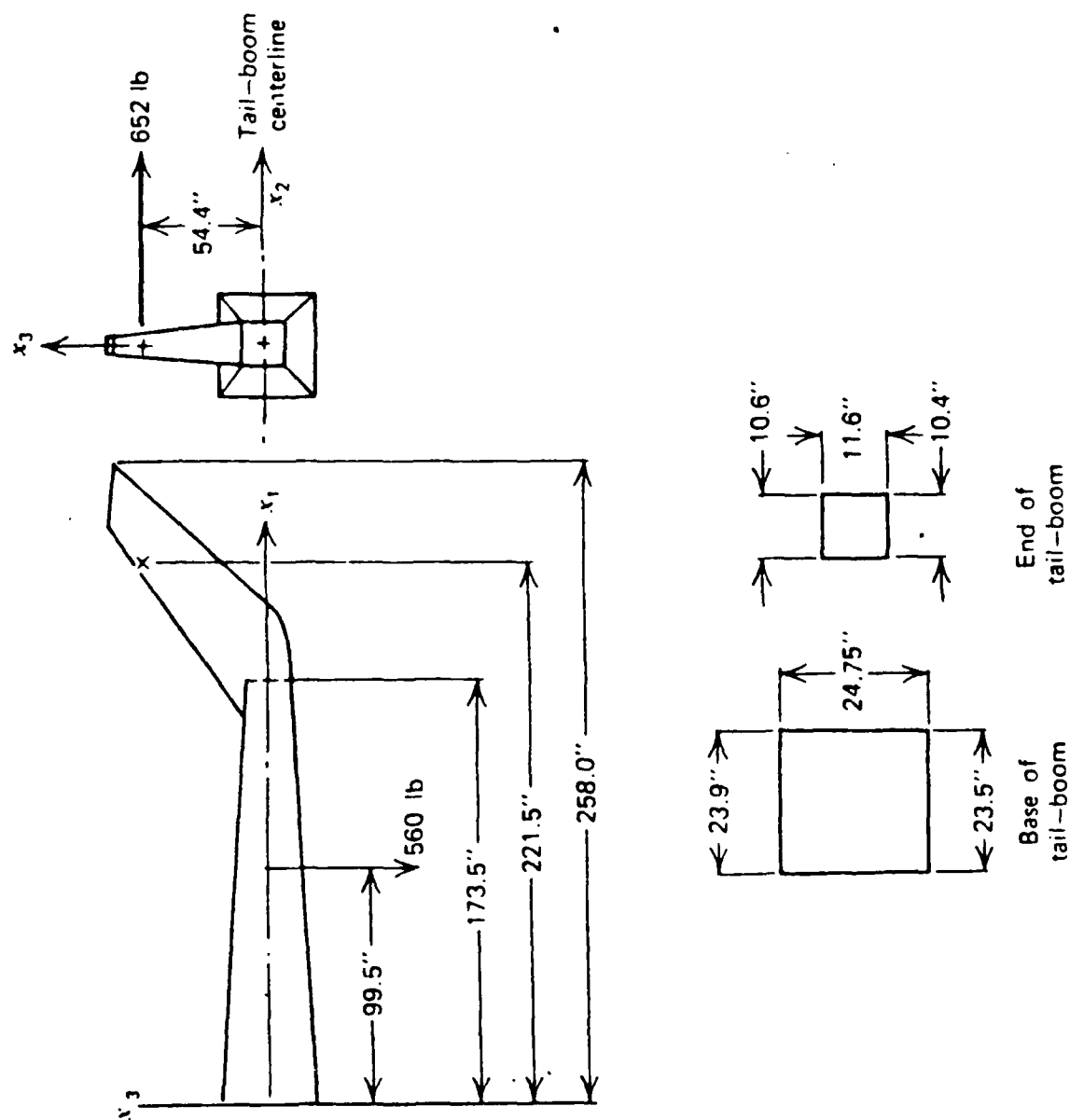
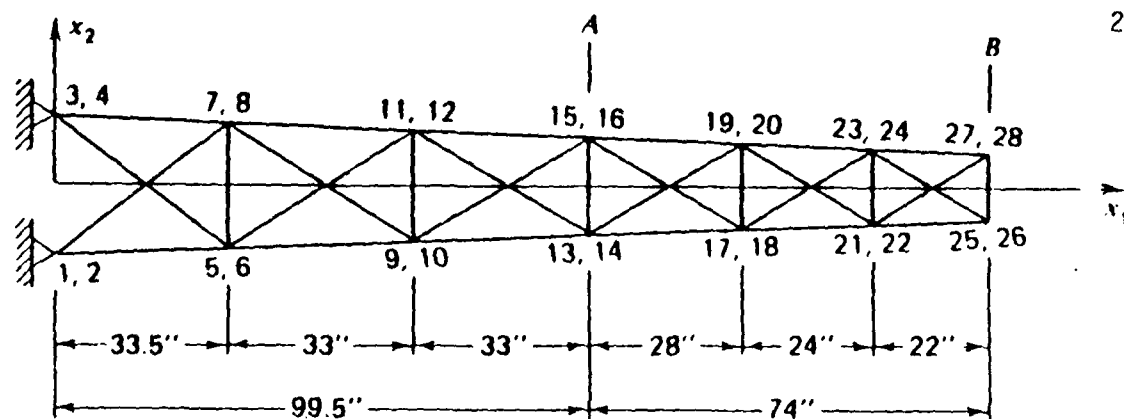
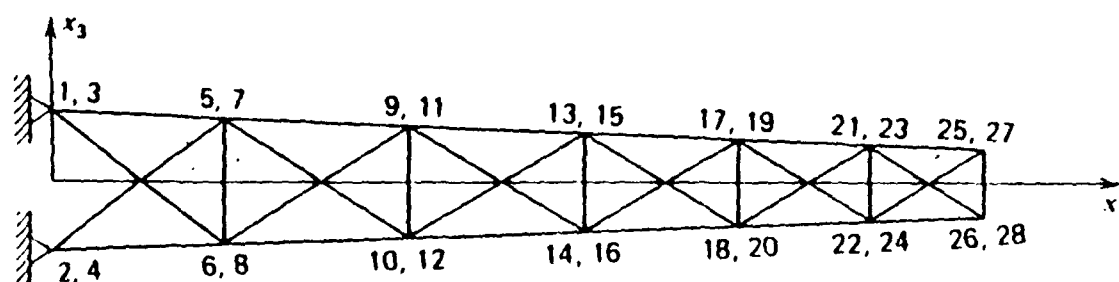


Figure 8.4.6 Geometry of Helicopter Tailboom





Top view



Front view

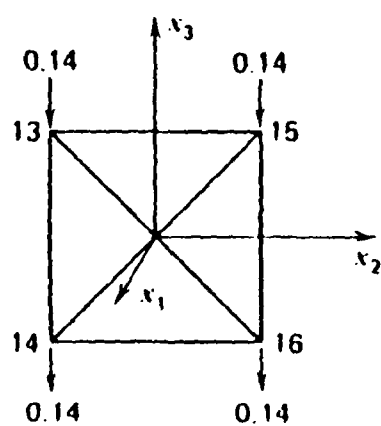
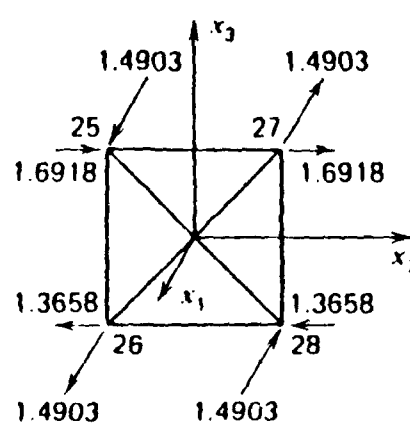
Loads at  
Section ALoads at  
Section B

Figure 8.4.7 Arrangement of Members for Open Truss-Helicopter Tailboom



## CHAPTER 9

### EVALUATION OF DATABASE, MIDAS AND COMPUTER PROGRAM FOR STRUCTURAL DESIGN OPTIMIZATION

#### 9.1 Introductory Remark

In this chapter, the database, the database management system - MIDAS, and the computer program for structural design optimization are evaluated. This evaluation is intended to bring out suitability and drawbacks associated with the use of the database and the database management system in the computer-aided structural design optimization environment. Note that the suitability and drawbacks of various database management concepts have been already discussed in Chapter 3. Also, the methodology developed to design a database was evaluated in Section 7.6. Evaluation of the database used in the structural design optimization program is given in Section 9.2. The database management system - MIDAS is evaluated in Section 9.3. The MIDAS/R data definition, data manipulation and query language of MIDAS are evaluated. Performance of MIDAS in engineering applications is given. Finally the performance of the computer program for structural design optimization is discussed.

#### 9.2 Evaluation of the Database Used in the Program

The design of the database and the computer program were discussed in Chapters 7 and 8, respectively. Several criteria are



selected to evaluate the database. They are (i) Simplicity, (ii) Ease of use, (iii) Ability to represent structural design data, (iv) Data redundancy, (v) Data consistency, (vi) Efficiency of data access, (vii) Ability to satisfy data needs of future applications, (viii) Ability to store additional data, (ix) Accommodation of various data types, (x) Representation of large vectors and matrices, (xi) Iterative data organization, (xii) Data independence, (xiii) Ease of database design, and (xiv) Implementational requirements.

**Simplicity.** The computer program used the simple tabular structure of the database which was designed using a relational model. The general tabular form of data such as element connectivity, nodal coordinates, and material property, was found simple to organize using the relational data model. The program was able to organize large matrices and vectors in a simple way using the numerical data model. The simplicity of data organization was realized from the point of view of users of the database.

**Ease of Use.** The database was found easy to use in the structural design optimization application program. Rows of relations were stored and retrieved easily. Large vectors and matrices were stored and retrieved easily in row or submatrix order.

**Ability to Represent Structural Design Data.** The computer program used the database to store all the data used and generated in the finite element analysis, design sensitivity and design change computations. The database was able to represent both the tabular and matrix type of



data used in the computation. Tabular type of data was appropriately represented in the database using the relational data model. Large matrices and vectors were suitably represented in the database using the numerical data model. However, it was found that the database design was not adequate to accommodate some particular data, such as intermediate data needed for assembly of large stiffness and load matrices. This was because, these types of data were not included while designing the conceptual and internal model of the database. Examples of intermediate data are - element numbers contributing to submatrices of assembled stiffness matrix, and load case numbers corresponding to load submatrices. In such cases, additional relations were added at the implementation stage to store the intermediate data. Data belonging to a structure, substructures, elements and nodes was represented using the relational data model. Even though, the relational data model was able to accommodate these data, the model did not take advantage of the hierarchical pattern of data. The relational data model accommodated, for example, data belonging to substructures by assigning substructure numbers to the relation names. Representation of data belonging to different loading cases was possible by including an additional attribute in relations corresponding to loading case numbers. However, this was found inconvenient to use, since it necessitated use of pointers to rows to identify data of a particular load case. In general, most of the data used by the computer program was represented in the database.



**Data Redundancy.** It was found that the data redundancy in the database was minimum because of the rigorous methodology used while designing the database. In some instance, data redundancy occurred because of the addition of new relations during the implementation stage of the computer program. New relations had to be included in the database either because of faulty database design or to minimize number of access required to retrieve the data. For example, it was necessary to include a relation EPRD containing data derived from relations ELVEC and NODVEC to minimize number of accesses to the database. In another instance, when only one load case is used in the computations, then load case attribute in relations ELRES, SUBVEC and STRVEC become redundant. Therefore, it was realized that it was impossible to avoid total data redundancy in the database. But, the database as such satisfied the requirement of minimum data redundancy.

**Data Consistency.** To a large extent, the data in the database was stored consistently. Some inconsistency in data may still occur because of the redundant data in the database. As explained earlier, redundancy in the database was present due to addition of new relations such as EPRD. To see if the database is consistent, consider modification of the relation EPRD due a change in cross-sectional property. data. If cross-section data is changed in this relation and if corresponding changes are not made in the relation ELVEC containing the same data, then the data in the database becomes inconsistent. To overcome this problem, the application program was designed to keep track of the redundant data and make proper changes in the required relations.



**Efficiency of Data Access.** The computer program for structural design optimization was able to access the data required for computation to minimize the in a minimum number of accesses. It was possible to minimize the number of accesses to database by designing various relations using the methodology discussed in Chapter IV. For example, one access to database was sufficient to retrieve data of element displacements. With two access to the database, material property of a particular element was obtained. In this case, first access retrieved the material number used by an element, and the second access retrieved the actual material property values. Again, two accesses were required to get data for assembly of stiffness matrix. Here, one access was required to obtain element degree of freedom numbers and another access to get element stiffness matrix. Assembly process was made more efficient by not storing element stiffness matrices in the database, thereby reducing the number of accesses to the database to only one. Element stiffness matrices were generated as and when they were needed. This computation of element stiffness matrices needed data from three different relations. For example, to get coordinates of nodes and material property of a particular element, access was needed to the relation containing node numbers and material number of the element; to the relation containing coordinates of nodes; and material property values. For an  $n$  node element,  $n$  accesses to relation containing coordinates of data were necessary. To increase the efficiency of accessing data for stiffness computation, element nodal coordinates and cross-sectional properties were stored together at the expense of data



redundancy. In case of large matrices, the database design, however, lacks efficiency in accessing a matrix data in column order which was stored in a row order or vice versa. This kind of operation requires a very large number of accesses to the database. In general, the database was able to provide data in an efficient way.

**Ability to Satisfy Data Needs of Future Applications.** The relations used in the database are capable of expanding both horizontally and vertically to satisfy the needs of future applications. Addition of new attributes at the end of existing attributes in a relation brings about horizontal growth of a relation. Addition of new relations will provide vertical growth of the database. For example, consider the need to use the database for nonlinear analysis. Assuming that the existing relation ELPAR does not have an attribute, for example nonlinearity index, then this attribute is added into the relation. By this method, both the existing and the new application program will be able to use the same database. This addition of new attributes does not require any modification of existing application using the relation. This addition of a new attribute which is associated with a particular entity (in this case-element) is possible only when the implemented relation has this entity key; otherwise, a new relation is added to the set of existing relations to make the database grow vertically. Thus, we see from the above discussion, that the database has capability to satisfy the data needs of future applications.



**Ability to Store Additional Data.** The relations used in the database have almost no limitation in storing any number of rows of data. Also, the relations have the capacity to store data as and when they are created during various stages of computation. This capability was achieved since the database uses hash addressing scheme to specify storage location of rows of various relations. But, it was found that this scheme has disadvantage in not locating particular rows of a relation in minimum number of accesses. Later, in Section 9.4, it will be shown that the number of accesses to the physical database is very high using this scheme. Further, for a particular implementation of relations in the database, number of attributes in a relation is fixed. Therefore, it is not possible to add additional attributes after a relation has already been defined using DDL. But, by modifying data definition call statements, new attributes can be added easily.

.

**Ability to Store Different Data Types.** In the computer program for structural design optimization, several different data types - integer, real, double precision and character words were used. Data belonging to these data types were stored in relations. Attributes of the relations in the database used these data types in many different combinations. Therefore, the database has the ability to store different data types.

**Ability to Store Large Vectors and Matrices.** The database was able to store large matrices and vectors needed for computation. Systematic organization of large matrices and vectors was possible by using the numerical data model. Vectors were stored in row order. Matrices were



stored submatrix order. The database was able to accommodate variable length vectors. Database, however, does not have the capability to store vectors in column order. The size of a row is limited to 1024 words. Number of elements in a submatrix cannot exceed 1024 words. The database design, also lacks the capability to avoid storage of zero values within a row, or submatrix.

**Ability to Organize Iterative Data.** The database has the flexibility to accommodate iterative data of structural design optimization. At the implementation stage of the database, the relations and matrices were classified into two groups, one group containing data which remained unaltered at various iterations, and the second group whose data changed in each iteration. Where data of previous iteration was not needed they were deleted at the beginning of the current iteration. The concept of global and local databases was used to support the iterative data.

**Data Independence.** The physical storage structure of the database is transparent to the user. The user or application program operates only on the logical database. The physical storage structure can alter without affecting the application program. Hence, data independence is achieved in the database design.

**Ease of Database Design.** Even though, the conceptual data model was found tedious to design, it was found very useful to design relations in the database. Relations for the internal data model were easy to construct simply by specifying various attributes in a



relation. Also, modification of one relation generally did not force reorganization of other relations that were already designed. It was found easy to add new relations into the database, since no pointers or links were needed to other relations. Also, large matrix data organization using numerical data model was found easy to incorporate in the database.

**Implementational Requirement.** The database design did not impose any special restrictions on the use of a particular database management system. Even though, the database for the structural design optimization program was implemented using MIDAS/R, it is also possible to use MIDAS/N.

### **9.3 Evaluation of DDL, DML and Query Language of MIDAS**

The data definition, data manipulation and query languages of MIDAS/R were used in the computer program for structural design optimization. The subroutines needed for using these languages were described in Section 6.2. These subroutines/commands are evaluated here to see if they satisfied the requirements specified in Sections 5.3 to 5.5. This evaluation brings out the suitability and drawbacks in using these languages in computer-aided structural design application. the evaluation is based on the experience in using these languages in the computer program described in Chapter 8.

**Evaluation of DDL of MIDAS/R.** The computer program used the data definition language of MIDAS/R to define various relations and matrices



needed for computation. Subroutines of DDL were found simple to use. MIDAS/R could define and use only one database at a time. The database and relation names had to be less than 6 and 8 characters, respectively. It did not impose any restrictions in defining any number of relations in a database. There was some restriction in the definition of the length of a row of a relation which was limited to 1024 words. Because of this, large vectors and matrices could not be accommodated in a relation. This problem was overcome to a certain extent by the use of submatrix approach to store data of large matrices. The language satisfied the requirements of dealing with various data types integer, real, double precision and character words. Since the number of rows in a relation is not required to be specified during data definition, MIDAS/R provided flexibility in allowing any number of rows in a relation. The DDL did not provide any facility to define relations and matrices for several substructures in the same database. Because of this, same relation and matrix names could not be used for data definition of substructures in one database. This problem was overcome by concatenating substructure number with the relation or matrix name. Redefinition of attributes in a relation, necessitated definition of a new relation and copying portion of data from the existing relation to the new relation. The DDL did not provide any facility to allow the user to specify number of pages and page size for efficient utilization of available memory. Since the application programmers generally have the knowledge about particular relations and matrices needed at various stage of



computation, control in deciding the page replacement should be made available to the application programmers. Other than some of the restrictions discussed above, the DDL satisfied the requirements specified in Section 5.3.

**Evaluation of DML of MIDAS/R.** The computer program for structural design optimization used the data manipulation language of MIDAS/R. The DML provided a set of subroutines which were useful and convenient in storing, retrieving, modifying and deleting data in the database. The language satisfied the requirements specified in Section 5.4. Several points are noted after using the DML in the computer program. MIDAS/R could only operate on one database at a time. If several databases are needed then they have to be used one after another by opening and closing the databases. RDSGET and RDSPUT routines operate on one row at a time. Even though, these routines have a less number of arguments, some penalty has to be paid in terms of efficiency when several successive rows have to be stored or retrieved. From this point of view, it is better to have starting and ending row numbers in the arguments of these subroutines to minimize number of calls to the database. The DML of MIDAS/R requires a call to RDSGET before actually modifying the data using RDSMOD. This was found inconvenient to use.

**Evaluation of Query Language.** The query language of MIDAS/R served two purposes. First, it was found very useful in developing and debugging the computer program for structural design optimization. Secondly, the query language provided a simple set of commands to



interactively define and manipulate the database. While developing and debugging the application program, query commands of MIDAS/R such as SELECT, LIST, CHANGE and DELETE were used to see if the application program has correctly defined and manipulated a database using the subroutines for DDL and DML. Sometimes, the incorrect data stored in a database due to erroneous application program logic was rectified using the query commands. Display of assembled stiffness submatrices on the terminal was sometimes useful in checking the correctness of the assembly procedure. Query language of MIDAS/R was found very useful for pre- and post-processing of data. Commands like 'SELECT STRESS GT 100.0', 'SELECT DISPLACE GT 10.0 AND LT 100.0', 'SELECT COORD-X FROM NODE DATA' was found very useful. Query language provided a simple set of commands to see the contents of the database. The language satisfied the requirements specified in Section 5.5.

#### **9.4 Evaluation of MIDAS in Structural Design Applications**

In this section, evaluation of MIDAS in structural design applications is made. Out-of-core linear simultaneous equation solvers are developed to evaluate the performance of MIDAS. Two types of equation solvers are developed, one using skyline approach and another using hypermatrix approach. First, the performance of MIDAS/N and MIDAS/R are evaluated using skyline approach. Later, they are evaluated using hypermatrix approach. This evaluation brings out several aspects such as efficiency of MIDAS/N and MIDAS/R in equation solving environment, suitability of skyline and hypermatrix approach for large



equation solving applications, use of memory management in engineering application, and design aspect of structural optimization program. In the following subsections and details of evaluation are given.

#### **9.4.1 Skyline and Hypermatrix Approaches**

Computer programs for solving equations using skyline and hypermatrix approaches are developed. The skyline approach uses left hand side coefficient matrix in the skyline form. Each column of the LHS coefficient matrix is of variable length storing only nonzero values. Each column, however, includes zero values within it. The right hand side and solution for unknowns are stored in a column vector. Numerical data model consisting of two levels of matrix data organization is used. First level contains details about skyline height and addresses of diagonal elements and the second level contains actual matrix data. Two separate programs are developed for solving equations, one using MIDAS/N and the other using MIDAS/R system.

Computer program for solving equations using hypermatrix approach use LHS coefficient matrix in the hypermatrix form. Large matrix is divided into number of submatrices. Only upper symmetric portion of LHS matrix is stored. Similarly the RHS matrix is also divided into a number of submatrices. The solution vectors are also stored in submatrix form. Null submatrices in LHS and RHS hypermatrices are not stored and manipulated. Partially full submatrix, however, stores zero coefficients within it. Again, numerical data model is used for matrix data organization. First level, contains nonnull submatrix numbers and



their sizes, and second level contains actual matrix data. Another two programs are developed; one using MIDAS/N and other using MIDAS/R system.

Both the approaches use Gaussian elimination scheme to decompose the coefficient matrix. Forward and backward substitutions are carried out to obtain solution of equations.

Thus, totally four computer programs are developed for solving equations. Program SKYSOL solves equations using MIDAS/R by the skyline approach. Program SKYSOLB (Shyy, 1985) solves equations using MIDAS/N by the skyline approach. Program HYP SOL uses MIDAS/R to solve equations by hypermatrix approach. Program HYPMDN uses MIDAS/N to solve equations by the hypermatrix approach.

Two sets of large equations are solved using the four computer programs SKYSOL, SKYSOLB, HYP SOL and HYPMDN. First set of equations has 1000 unknowns and half band width of 10. Second set of equations has 1000 unknowns and half band width of 30. For hypermatrix approach submatrix size of 10 is used. These equations are solved with different memory management schemes. Performances of these programs are measured by noting central processing unit time (CPU), number of reads made on physical database (NREAD), number of writes made on physical database (NWRITE), and number of calls made to database management subroutines (NCALL).

The details of performance of these programs are given in subsequent sections.



### 9.4.2 Performance of MIDAS/R

The two computer programs SKYSOL and HYP SOL using MIDAS/R were used to solve the two sets of equations. The results of CPU, NREAD, NWRITE and NCALL are given in Table 9.4.1. The table shows the results for different cases of page size and number of pages. Three different cases (i) 20 pages of 256 short integer words, (ii) 20 pages of 1024 short integer words, and (iii) 2 pages of 20480 short integer words are used.

The following points are observed from the table. As the page size is increased CPU time for solving equations using skyline and hypermatrix approaches reduces. Number of reads (NREAD) for large page size is less than for small page size. NWRITE also had similar trend except for the case of hypermatrix approach with page size of 1024 words, where NWRITE increased when page size is increased. The number of writes is much smaller than number of reads. This may be due to the hash index scheme used in the MIDAS/R addressing and searching routines. The MIDAS/R program makes a huge number of reads to search and locate the required data. Further, note that number of calls to MIDAS/R routines remained same for all page sizes, as expected. Number of calls to MIDAS/R increased as bandwidth increased.

Skyline approach for solving first set of equations with a page size of 256 words took 228.55 seconds, whereas hypermatrix approach for solving same equations with same memory management scheme took 2720.87 seconds. The hypermatrix approach is about 10 times slower than the skyline approach. In solving the second set of equations with bandwidth of 30, skyline approach used 3138.66 CPU seconds, whereas



Table 9.4.1 Performance of MIDAS/R

Number of equations : 1000  
 Half band width : 10

Page Size	Number of Pages	Skyline Approach				Hypermatrix Approach			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	228.55	16103	218	12984	2720.87	92312	564	12082
1024	20	119.00	13970	74	12984	1623.00	50678	619	12082
20480	2	50.75	303	65	12984	928.00	26644	495	12082

Number of equations : 1000  
 Half band width : 30

Page Size	Number of Pages	Skyline Approach				Hypermatrix Approach			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	3138.66	65602	530	32594	8207.61	275060	1157	24117
1024	20	1576.16	31417	191	32594	3717.68	115788	1327	24117
20480	2	1297.96	27481	145	32594	1311.56	27497	2598	24117



hypermatrix approach used 8207.61 CPU seconds. In this case, the hypermatrix approach is about 3 times slower than the skyline approach. Thus, we see from the results that as band width is increased the difference in CPU time between skyline and hypermatrix approaches reduces.

#### 9.4.3 Performance of MIDAS/N

The two computer programs SKYSOLB (Shyy, 1985) and HYPMDN using MIDAS/N were used to solve the two sets of equations. The results of CPU, NREAD, NWRITE and NCALL are given in Table 9.4.2. The equations are solved for different cases of page size and number of pages. Two different cases (i) 20 pages of 256 short integer words, and (ii) 20 pages of 1024 short integer words.

The following points are observed from the table. As the page size is increased from 256 words to 1024 words, the CPU time for solving equation using skyline and hypermatrix approaches reduces, but not to a considerable extent. Number of reads and number of writes for large page size is much less than for small page size. Number of writes is smaller than number of reads. This is due to replacement of unmodified pages by new data causing more reads than writes. Note that number of calls to MIDAS/N routines remains same for all page sizes as expected. Number of calls to MIDAS/N increased as band width increased.

Skyline approach for solving equations of 1000 unknowns with a half band width of 10 and page size of 256 words took 33.05 seconds, whereas hypermatrix approach for solving same equation with same memory



Table 9.4.2 Performance of MIDAS/N

Number of equations : 1000  
 Half band width : 10

Page Size	Number of Pages	Skyline Approach				Hypermatrix Approach			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	33.05	340	127	16262	248.10	22698	1001	11487
1024	20	30.69	70	30	16262	211.40	6013	339	11487

Number of equations : 1000  
 Half band width : 30

Page Size	Number of Pages	Skyline Approach				Hypermatrix Approach			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	125.36	1394	480	38688	486.02	39545	1326	22939
1024	20	114.73	337	120	38688	411.42	10370	435	22939



management scheme took 248.10 seconds. The hypermatrix approach is about 8 times slower than the skyline approach. In solving the second set of equations with a band width of 30, skyline approach used 125.36 seconds, whereas hypermatrix approach used 486.02. In this case, hypermatrix is about 4 times slower than the skyline approach. We see from the result that as band width increases the difference in CPU time between skyline and hypermatrix approaches reduces. The reason being, percentage of nonzero elements within the nonnull submatrices considerably reduces as band width increases in the two sets of equations. Arithmetic operations on nonzero elements reduces with less percentage of nonzero elements bringing about reduced CPU time.

#### **9.4.4. Comparison of MIDAS/N and MIDAS/R Using Skyline Approach**

In this section, the efficiency of MIDAS/N and MIDAS/R is compared in solving equations using the skyline approach. The two computer programs SKYSOL and SKYSOLB were used to solve the two sets of equations. The comparison of results is given in Table 9.4.3. Equations are solved for different cases of page size and number of pages. Two different cases (i) 20 pages of 256 words, and (ii) 20 pages of 1024 words are used for comparison.

The following points are observed from the table. MIDAS/N used 33.05 seconds of CPU time for solving 1000 equations with half band width of 10 and page size of 256 words, whereas MIDAS/R for solving the same equations took 228.56 seconds. Thus, we see that MIDAS/R is about 8 times slower in this case. For page size of 1024 words, MIDAS/R is



**Table 9.4.3 Comparison of MIDAS/N and MIDAS/R Using Skyline Approach**

Problem details:

Number of equations : 1000  
 Half band width : 10  
 Skyline height : 10

Page Size	Number of Pages	MIDAS/N				MIDAS/R			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	33.05	340	127	16262	228.55	16103	218	12984
1024	20	30.69	70	30	16262	119.00	13970	74	12984

Problem details:

Number of equations : 1000  
 Half band width : 30  
 Skyline height : 30

Page Size	Number of Pages	MIDAS/N				MIDAS/R			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	125.36	1394	480	38688	3138.66	65602	530	32594
1024	20	114.73	337	120	38688	1576.16	31417	191	32594



about 4 times slower than MIDAS/N. The number of writes in MIDAS/R is about 2 times more than in MIDAS/N, and the number of reads in MIDAS/R is very high compared to MIDAS/N. We could attribute this high value of reads to the nature of addressing and searching used in MIDAS/R. MIDAS/N uses indexing scheme to locate the data whereas MIDAS/R uses hashing scheme which uses a large number of reads and searches to locate the required data. Number of calls to MIDAS/N, on the other hand were higher than in MIDAS/R. This is because, MIDAS/N uses addresses of diagonal elements to locate the columns of a skyline matrix which is stored in a single row vector. MIDAS/N is required to locate this address before accessing the skyline vector. MIDAS/R does not require such addresses, since skyline vectors are stored in variable length rows, thereby using less number of calls to MIDAS/R. In the case of the second set of equations with half band width of 30, similar trends of CPU, NREAD, NWRITE and NCALL are observed.

Therefore, from the above results, we see that MIDAS/R is inefficient compared to MIDAS/N.

#### **9.4.5 Comparison of MIDAS/N and MIDAS/R Using Hypermatrix Approach**

Here, the efficiency of MIDAS/N and MIDAS/R is compared in solving equations using the hypermatrix approach. The two computer programs HYP SOL and HYPMDN were used to solve the two sets of equations. The comparison of results is given in Table 9.4.4. Equations are solved for different cases of page size and number of pages.



**Table 9.4.4 Comparison of MIDAS/N and MIDAS/R Using Hypermatrix Approach**

Problem details:

Number of equations : 1000  
 Half band width : 10  
 Submatrix size : 10\*10

Page Size	Number of Pages	MIDAS/N				MIDAS/R			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	248.10	22698	1001	11487	2720.87	92312	564	12082
1024	20	211.40	6013	339	11487	1623.00	50678	619	12082

Problem details:

Number of equations : 1000  
 Half band width : 30  
 Submatrix size : 10\*10

Page Size	Number of Pages	MIDAS/N				MIDAS/R			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	486.02	39545	1326	22939	8207.61	275060	1157	24117
1024	20	411.42	10370	435	22939	3717.68	115788	1327	24117



The following points are observed from the table. MIDAS/N uses 248.10 seconds of CPU time to solve 1000 equations with half band width of 10 and page size of 256 words, whereas MIDAS/R takes 2720.87 seconds of CPU time to solve the same set of equations. Thus, MIDAS/R is about 10 times slower than MIDAS/N in this case. For page size of 1024 words, MIDAS/R is about 8 times slower than MIDAS/N. Number of reads in MIDAS/R is about 4 times more than in MIDAS/N for page size of 256 words. On the other hand, number of writes in MIDAS/R is about 2 times less than in MIDAS/N for the same size. For page size of 1024 words, number of reads in MIDAS/R is about 8 times more than in MIDAS/N, whereas number of writes in MIDAS/R is 2 times more than in MIDAS/N. Number of calls in MIDAS/R are slightly higher than in MIDAS/N. This is because, MIDAS/R uses 2 call statements - RDSGET and RDSMOD to modify data whereas MIDAS/N uses only one call statement RDSPUT to modify data. Similar trends of CPU, NREAD, NWRITE and NCALL are observed for the second set of equations.

Therefore, from these results we can see that MIDAS/R is inefficient compared in MIDAS/N.

#### **9.4.6 Performance of Memory Management of MIDAS/R**

As mentioned in Chapter 6, the MIDAS/R program is based on modification and extension of the RIM program. A new memory management interface was added to assess improvements in performance of the RIM program. Here, a comparison of performance of RIM's built-in memory management system and the performance of MIDAS/R with the additional memory management is made. The CPU times to solve 1000 equations are



used to compare the performance. Table 9.4.5 shows the CPU time for various size of RIM's built-in memory management scheme and additional memory management scheme. Observe from the table that the CPU time does not vary considerably by increasing the total memory of the built-in memory management scheme of RIM. This indicates that the RIM program is not capable of utilizing large memory of the computer effectively. By introducing the additional memory management scheme in MIDAS/R we see that CPU time reduces considerably. As page size increased from 256 words to 2048 words, CPU reduced from 228.55 seconds to 79.00 seconds for RIM's built-in memory of 9216 words. Similar trends are observed for other cases of memory sizes. Therefore, we see that memory management plays a very important role in a dynamic environment of engineering applications. Also, memory management of the RIM program can be substantially improved.

#### 9.5 Evaluation of the Computer Program for Structural Design Optimization Using MIDAS

The computer program for structural design optimization described in Chapter 8 is evaluated here. Evaluation of the program is based on several criteria (i) capability to analyze and optimize large structural systems, (ii) modular program features, (iii) element library, (iv) out-of-core solution schemes, (v) well designed database, (vi) database management system, and (vii) performance of the computer program. The general purpose capability of the program, modular program features, element library and out-of-core solution schemes used in the program are already described in Chapter 8. Based on the evaluation



Table 9.4.5 Performance of Memory Management of MIDAS/R

Number of equations : 1000  
 Half band width : 10

Total Memory of RIM's Built-in M.M	CPU	With Additional M.M. Interface		CPU
		Page Size	No. of Pages	
9216	1597.33	256	20	228.55
		1024	20	119.86
		2048	20	81.99
20480	1542.53	256	20	228.11
		1024	20	115.63
		2048	20	79.50
81920	1514.00	256	20	221.57
		1024	20	112.31
		2048	20	74.51
163840	1607.69	256	20	218.01
		1024	20	115.72
		2048	20	80.95



criteria (i) to (iv), the program is well designed and satisfied the requirements. Also, the program uses well designed database and database management system discussed in Chapter 7 and 6, respectively. Therefore, no further discussion on criteria (i) to (vi) is made here. Detailed evaluation of the performance of the computer program is based on several example problems described in Section 8.4.

The performance of the program is measured by noting parameters such as CPU time, number of reads and writes made on physical database, and number of calls made to the database management system. These parameters are noted while solving problems (i) ten bar truss (ii) twenty-five bar truss, (iii) forty-seven bar truss, (iv) seventy-two bar truss, (v) one hundred-eight bar truss, and (vi) two hundred bar truss. The data of these parameters are collected and given in Tables 9.5.1 to 9.5.6. Tables show these parameters for items - input and preprocessing step (1), finite element analysis (2), design sensitivity analysis (3), and design change computation done by the program IDESIGN3 (4). The tables show the CPU, NREAD, NWRITE and NCALL for both the total run for all iterations required for optimization and for each iteration of the optimization computation. Also, the tables show the total - CPU, NREAD, NWRITE and NCALL for the complete optimal design computation. The CPU time to solve the problems using the computer program are compared with the results given by Thanedar, Park and Arora 1985.

The following points are observed from these tables. The CPU required to solve the various problems using the present computer



Table 9.5.1 Summary of Evaluation Parameters for Ten Bar Truss

Item No.	For Final Solution				For each Iteration			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
1	18.23	352	106	419	18.23	352	106	419
2	15.03	165	22	2299	1.36	15	1	209
3	18.42	220	121	1419	1.67	10	11	129
4	34.61				3.14			
Total	86.29	737	249	4137				

CPU time from Thanedar, Park and Arora (1985): 5.5 sec.

Table 9.5.2 Summary of Evaluation Parameters for Twenty Five Bar Truss

Item No.	For Final Solution				For each Iteration			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
1	18.37	356	108	1005	18.67	356	108	1005
2	34.37	315	150	7365	2.29	21	10	491
3	32.89	390	390	4199	2.19	26	26	280
4	52.27				3.48			
Total	137.9	1061	648	12569				

CPU time from Thanedar, Park and Arora (1985): 7.4 sec.



**Table 9.5.3 Summary of Evaluation Parameters for Forty Seven Bar Truss**

Item No.	For Final Solution				For each Iteration			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
1	21.46	359	122	2405	21.46	359	122	2405
2	20.08	174	90	5862	3.34	29	15	977
3	47.52	312	426	7980	7.92	52	71	1330
4	51.30				8.55			
Total	140.36	845	638	16247				

CPU time from Thanedar, Park and Arora (1985): 64.0 sec.

**Table 9.5.4 Summary of Evaluation Parameters for Seventy Two Bar Truss**

Item No.	For Final Solution				For each Iteration			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
1	22.14	360	127	4862	22.14	360	127	4862
2	83.99	527	340	25483	4.94	31	20	1499
3	56.25	424	696	11208	5.11	38	63	1018
4	69.63				4.97			
Total	231.01	1331	1163	41553				

CPU from Thanedar, Park and Arora (1985): 29.3 sec.



**Table 9.5.5 Summary of Evaluation Parameters  
for One Hundred Eight Bar Truss**

Item No.	For Final Solution				For each Iteration			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
1	25.96	397	136	9107	25.96	397	136	9107
2	226.91	1320	870	67620	7.56	44	29	2254
3	379.84	1610	3128	60352	12.66	53	104	2011
4	230.75				7.69			
Total	863.46	3327	4134	137079				

**Table 9.5.6 Summary of Evaluation Parameters for Two Hundred Bar Truss**

Item No.	For Final Solution				For each Iteration			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
1	44.97	601	166	44081	44.97	601	166	44081
2	412.41	2070	1530	147870	13.74	69	51	4929
3	1210.21	27450	9810	135960	40.34	915	327	4532
4	330.00				11.00			
Total	2000.5	30121	11506	327911				



program and the one used in the reference are much different. The program used in Thanedar, Park and Arora (1985) performs finite element analysis and design sensitivity computation using only the available memory in the computer. No out-of-core data storage is used by the program. Whereas the computer program developed here uses the database and database management system described earlier. Therefore, the CPU time taken to solve these problems are high compared with the results given in the reference, as expected. For instance, CPU time of 86.29 seconds is used by the program compared to 5.5 seconds used by the program given in the reference. Similar trends are observed from the table for other problems.

Several other points are observed from the table. CPU time, NREAD, NWRITE and NCALL increase as the size of problem increase as expected. CPU time, NREAD, NWRITE and NCALL for input and preprocessing of data is small compared to those of finite element analysis, and design sensitivity computation. However, these values are very high compared to one iteration of finite element analysis and design sensitivity computation. One of the reasons is extensive preprocessing of data done before assembly of large matrices and solution of equations. This preprocessing has enabled us to achieve efficient assembly of matrices and solution of equations. Another reason, can be attributed to the use of hypermatrix scheme in the finite element analysis and design sensitivity computation. The hypermatrix scheme uses large amount of intermediate information such as element numbers contributing to submatrices, nonnull partition numbers for assembly and other computations. It requires lengthy programming logic and use of database



to generate such data. Thus, the input and preprocessing step consumes more CPU time to generate the required data.

Further, note that as number of calls made to database management system increases, so does the CPU time. This indicates that DBMS consumes enormous CPU time which is much higher than the CPU time required for actual computation. Therefore, it is very important to reduce the number of calls made to DBMS. Several ways to achieve this are (i) use as much incore data storage as possible, (ii) redesign relations so that number of accesses required to retrieve necessary data is minimum, (iii) include starting and ending row numbers in the data manipulation routines to store and retrieve relations, (iv) use adjoint sensitivity analysis method and (v) use skyline approach to design the computer program.



## CHAPTER 10

### SUMMARY, DISCUSSION AND CONCLUSIONS

#### 10.1 Summary

Computer-aided structural design means integration of structural engineering design methods and computer-science in a computer-based system containing a database, a program library and a man-machine communication link. With this definition in view, a new concept is described for integrating finite element analysis and design optimization methodology into a computer-based system. Emphasis is placed upon database management concept for structural design. Several reasons exist for emphasizing data management in design. First, the iterative nature of optimal design process uses large amount of data for computation. Secondly, existing finite element programs are not flexible to use modified or generated data at various stages of design. Thirdly, designer needs control over the program and data to obtain optimum design. Finally, a good database will enable addition of new of optimization and other programs without extensive modification of database or existing programs. Also, several designers can be allowed to use a common database to investigate alternate designs.

Structural design process is described to bring out various steps involved in design optimization. Mathematical modeling of the design process is presented to describe the nature of computation and data used



in design. Important components required to build a computer-aided structural design system are described. Need for a good data management system is emphasized. Users of computer-aided structural design system are identified and their requirements are described.

A study of database management concepts applicable to finite element analysis and structural design optimization is conducted. This study is essential since the data managements concepts are relatively new to engineering community. Definition of various terminologies is given and described with reference to examples from finite element analysis and optimization data. Hierarchical, network, relational and numerical data models are described. The advantages and disadvantages of these models are discussed. Relational and numerical data models are found to be appropriate for structural data organization. The concept of normalization of data is described. This concept provides certain guidelines to group different data items to form associations. Global and local database concepts are described and their use of design optimization is brought out.

A methodology to design a structural design database is proposed. Up until now, no such methodology has been used for data organization of existing finite element programs. No scientific database management techniques were available at the time those programs were developed. Three levels of data organization -- conceptual internal and external are suggested for structural design databases. Data organization at the conceptual level represents inherent characteristics of data regardless of whether or not database management software available



BD-A174 458

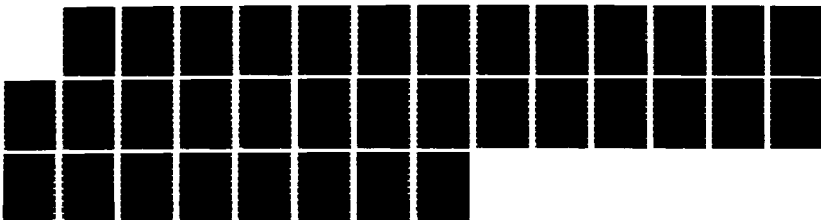
COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION USING A  
DATABASE MANAGEMENT (U) IOWA UNIV IOWA CITY OPTIMAL  
DESIGN LAB T SREEKANTAMURTHY ET AL 30 SEP 86

4/4

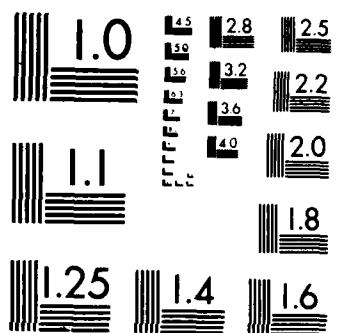
UNCLASSIFIED

ODL-85-17 AFOSR-TR-86-2069 AFFSR-82-0322 F/G 18/3

NL







MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A



supports such organization directly. Various steps are identified to develop a conceptual data model. Methodology for including vector and matrix data in the conceptual data model is described. A methodology for constructing an internal model is proposed. The internal model aims to store the structural design data in an efficient way. Considerations for reducing storage space and data access time are discussed. Normalization of data is suggested to avoid various anomalies in storage operation. A method for developing an external model is given. An external data model enables us to provide data to several application programs depending on their needs. One of the important aspects in the design of database for structural analysis matrix data. A methodology is developed to store large matrix data in the database. Various types of large matrices -- square, triangular, banded, hypermatrix, and skyline matrix -- are identified and their characteristics are studied. Various aspects like storage space, processing sequence, matrix operations, page size, flexibility of modification, etc., are considered to develop suitable storage schemes. A numerical model is developed to represent large order matrix data. Finally, an algorithmic model is proposed to deal with storage and computation efficiency aspects required for structural design optimization programs.

A proposal to develop a database management system for structural analysis and optimal design is made. Components required for a good database management system are identified. Some important components of the database management system are -- languages, command processors, addressing and searching, file definition and file operation, memory



management and security routines. Requirements of data definition and data manipulation languages are formulated. Query language is proposed for an interactive user of a database. A syntax (grammar) for these languages is given. A survey of existing database management systems is conducted. Requirements of a database management system are formulated.

Implementation of a database management system -- MIDAS was done. MIDAS program has two subsystems -- MIDAS/R and MIDAS/N. MIDAS/R is based on relational model of data and MIDAS/N (Shyy, 1985) is based on numerical model of data. MIDAS/R relieves the burden of managing data for application programmers by providing user-friendly application commands. The program has sophisticated interactive commands to query the database. Relations can be defined and manipulated using data definition and data manipulation commands of MIDAS/R. Interactive commands of MIDAS/R can also be used to define and manipulate relations in a database. MIDAS/R has capability to store any number of relations in a database. Numerical database management system -- MIDAS/N has capability to store matrix data of finite element analysis and optimization programs.

A database is designed to store data of finite element analysis and structural design optimization. The design is completed in three phases using the methodology developed. Conceptual data model is designed by identifying entities, entity keys and domains and forming elementary relations. Internal data model is designed based on needs of computation process and normalization procedures. This model is used for implementation of database. Also, some relations are suggested to



serve as an external data model. The methodology of database design is evaluated in view of the actual database design.

A computer program is developed for finite element analysis and design sensitivity analysis. The program has several modules which are functionally independent. The program uses the database and the database management systems - MIDAS/R. The program is capable of optimizing general structural design problems. Capabilities of the program -- element library, hypermatrix assembly, out-of-core solution of equations, imposing stress and displacement constraints are explained. The modules used in the program are described. A number of example problems are solved using the program.

Finally, the database, the database management system - MIDAS, and the computer program for structural design optimization are evaluated. The database is evaluated using several criteria. The data definition, data manipulation and query languages of MIDAS are evaluated. Computer programs for out-of-core solution of equations using skyline and hypermatrix approaches are developed. The performance of MIDAS/R and MIDAS/N is determined. The efficiency of both the systems are compared. The computer program for structural design optimization is evaluated.

## 10.2 Discussion

The study answers several problems facing data management in finite element analysis and structural design optimization computations. The following questions were addressed in the study: (i) how a database has



to be organized? (ii) what kind of information is to be stored? (iii) what kind of database management system is suitable? (iv) how data is manipulated? and (v) how various finite element analysis and design optimization applications use the data? Answers to these questions were not available, since only a few research studies had been made on the topic. Thus, we were faced with the problem of how to answer these questions. In this regard, sophisticated techniques were available to organize data of business applications. Our next question was, is it possible to use those techniques to organize data of engineering applications? Soon it was realized that data of business applications were of different nature than the data of engineering applications. But the question arose as to whether engineering data could be similarly modeled? If so, do the structural design databases require different considerations from database of business applications? These questions were not adequately answered previously. Several different approaches had to be taken; for example, a study of database management concepts, use of available database management systems, and development of special structural design database management systems.

Therefore, an attempt was made to study all the relevant data management concepts. The concepts that were found applicable were described with reference to examples for finite element analysis and design optimization data. The database management approach presented in the study offers solution to many of these problems. The study shows how data of finite element analysis and optimization can be organized using various data models. Relational and numerical models were found



to represent all the characteristics of structural design data and therefore they were selected for detailed study. Questions were posed as to how to decide what data items have to be grouped together? In particular, using a relational model, how do we determine what relations are needed? and what their attributes should be? It was shown that normalization of data provides certain guidelines to group the general data items. It was realized that the normal form of data did not suggest schemes to group data of large matrices. Further, the concept of global and local databases was found useful in organizing iterative data of structural design optimization.

After, we formulated the database management concepts for computer-aided structural design optimization, the next question was how to design a database? Is it possible to develop a methodology for designing a database for structural analysis and design? If possible how do we begin to develop a methodology? Are the methodologies used in business application suitable for our purpose? If not, what aspects should be considered in developing a methodology for structural design databases? A methodology was proposed considering several features and requirements of finite element analysis and structural design optimization applications. Methodology used relational and numerical data models for designing a database. Special consideration was given to incorporate large matrix data into the database. A database was designed using the methodology. This database was used for actual implementation of the computer program for structural design optimization. The methodology was evaluated to see if it was suitable



to design databases of structural optimization applications. It was realized that the methodology was to some extent dependent on the way in which characteristics of data are analyzed. Detailed analysis of entities, domains and functional dependencies were needed which was sometimes difficult to make. But, in general the methodology enabled us to design a well organized database for structural design application.

Later, in the study, we dealt with the need of a software for database management. What kind of database management program is suitable? Is it possible to use an existing DBMS or develop a new database management system? What modifications are required to existing DBMS so that it can be used in structural design application? These questions were tackled by a detailed study of requirements of a DBMS. It was realized that data definition and data manipulation languages play an important role in providing a *communication* like between designer and computer system. MIDAS program for database management was developed. The program was able to organize both relations and matrix data. This program was found to meet most of the requirements of a DBMS.

A finite element analysis and structural design optimization program was developed using the database and the database management system MIDAS. The program was able to solve complex structural design problems. Modular program design, element library, hypermatrix assembly scheme, out-of-core solution, stress and displacement constraint capabilities of the program were found very useful. The program was developed to find out the suitability of the database design, efficiency



of the database management system, and efficiency of the structural design optimization program. Also, several different types of equation solvers were developed using skyline and hypermatrix approaches incorporating MIDAS for out-of-core solution. It was realized that performance of DBMS needs careful consideration in designing a DBMS for engineering applications.

### 10.3 Conclusions

Important conclusions of the study are listed below.

1. The study has shown that computer-aided design optimization of complex structural systems can be attempted by integrating finite element-based-optimal design methods and computer-science methods into a computer-based system containing a database, a program library and man-machine communication link.
2. The study has shown that database management plays a very important role in developing a computer-aided design system. Several problems of data organization in finite element analysis and optimization applications can be overcome by providing sophisticated database management systems.
3. The study has shown that various database management concepts are applicable to data organization of structural design problems. Several concepts such as relational and numerical data model are useful to organize tabular and matrix type of data used in computation. Suitability and drawbacks of these concepts are explained.



4. The methodology developed for designing a database for structural design application is useful and this methodology will replace the intuitive way of data organization in finite element analysis and optimization programs. The methodology, to a certain extent, is dependent on the analysis of characteristics of the data. Such analysis of complex structural design data is not easy. Methodology to organize both tabular type of data and large matrix data was very useful. More study is needed to consolidate the methodology developed herein.
5. Formulation of requirements of a database management system has enabled us to develop and use proper database management system for structural design optimization. Formulation of requirements of data definition, data manipulation and query languages has helped in providing a good communication link between computer and application programmer.
6. A suitable database management system was implemented by selecting, modifying and extending an existing database management system to meet the requirements of structural design applications. This implementation has enabled us to explain suitability and drawbacks of existing DBMS.



7. The database designed using the methodology is capable of storing structural design data. The database is well organized and meets the requirements of finite element analysis and structural design optimization applications. This design of database has enabled us to demonstrate the use of database design methodology. Our objective of having a well designed database for computer-aided structural design was met.
8. Implementation of a finite element program and structural design optimization program using the database and database management system has shown that computer-aided design of complex structural systems can be attempted. It was realized such systems can be developed easily and quickly with the use of database and database management systems.
9. Evaluation of database management system has indicated that some compromise has to be made for the use of sophisticated DBMS in terms of inefficiency in computer CPU time. More study is needed to improve the efficiency of DBMS for engineering applications.

#### **10.4 Scope for Future Work**

This extensive study on database management for computer-aided structural design optimization has shown that there is great potential for future work. Several areas are suggested for extending this work.



1. Develop a more sophisticated database management system that is capable of dealing with relational and numerical data model simultaneously.
2. Develop a detailed conceptual data model by considering not only the basic association between data but also processing sequence of computation to obtain a complete theoretical analysis of structural design data.
3. Improve the efficiency of database management system by enhancing the memory management schemes.
4. Develop a methodology to integrate several existing finite element analysis programs into a common database. A more detailed study of external model is required.
5. Methodology for integrating several databases for existing finite element analysis programs needs to be developed. Network of databases appears to be very useful in engineering applications.
6. Study how a database can be utilized in parallel processing of data in structural analysis and design optimization.
7. Use of database and database management system in mini- and micro computer systems needs to be studied.
8. Use of database and database management system in finite element mesh generation and mesh display needs further study.



APPENDIX I  
AN ALGORITHM FOR TRANSITIVE CLOSURE



An algorithm for determining transitive closure from a given connectivity matrix  $C$  is as follows:

1. Form matrix  $J$  with

$$J(i,i) = 0 \text{ for } 1 \leq i \leq n$$

$$= 1 \text{ for } i \neq j$$

2. Form modified connectivity matrix  $CC$

$$\text{with } CC(i,j) = C(i,j) \quad 1 \leq i \leq n$$

$$1 \leq j \leq n$$

$$\text{and } CC(i,i) = 0$$

3. DO  $i = 1, n$

DO  $k = 1, n$

DO  $j = 1, n$

$$\text{If } (CC(i,j) \cdot EQ \cdot 1 \cdot AND \cdot CC(j,k) \cdot EQ \cdot 1) C(i,k) = 1$$

End DO loop

4. Remove erroneous dependencies that were derived from the situation

$$N_i \rightarrow N_j \rightarrow N_i$$

5. IS new matrix  $C$  obtained? If no, stop

6. Form modified connectivity matrix (as in step 2) using  $CC$  matrix derived in step 3

7. Go to step 3

8. End



**APPENDIX II**  
**BNF DESCRIPTION OF THE PROPOSED**  
**DATA DEFINITION LANGUAGE**



```

<letter>:: = A|B|C|.....|Z
<digits>:: = 1|2|.....9|0
<basic symbols>:: = <letters>|<digits>
<string>:: = <any sequence of basic symbols>
<variable>:: = <simple variable>|<subscripted variable>
<simple variable>:: = <identifier>
<identifier>:: = <letter>|<identifer><letter or digit>
<subscripted variable>:: = <identifer>[<subscript list>]
<subscript list>:: = <fortran subscript list>
<empty>:: = <null string of symbols>
<unsigned integer>:: = <digit>|<unsigned integer><digit>
<data definition statement>:: =
    <database definition statement>|
    <database user specification statement>|
    <database definition statement>|
    <relation data definition statement>|
    <numerical data definition statement>|
    <data definition termination statement>|
    <data redefinition statement>
<database definition statement>:: =
    <reserved procedure DBDEFN>
    (<database definition parameter part>)
<database definition parameter part>:: =
    <database name>
    <database hierarchy>

```



```

        <database status>
        <database definition error code>

<database name>:: = '<name>'<variable>
<database heirarchy>:: = '<heirarchy type>'<variable>
<hierarchy type>:: = GLOBAL|LOCAL
<name>:: = <letter><string>
<database status>:: = '<status type>'<variable>
<status type>:: = PERMANENT|TEMPORARY
<database definition error code>:: = <variable>
<database user identification statement>:: =
        reserved procedure DBUSER>
        (<database user identification parameter part>)
<database user identification parameter part>:: =
        <database name>,
        <database user type>,
        <database access type>, <password>,
        <user identification error code>

<database user type>:: = '<user type>'<variable>
<user type>: = OWNER|USER
<database access type>:: = '<access type>'<variable>
<access type>:: = READ|MODIFY
<password>:: = <letter><string>
<user identification error code>:: = <variable>
<data set definition statement>:: =

```



```

        <reserved procedure DSDEFN>
        (<dataset definition parameter part>)

<data set definition parameter part>:: =
        <database name>,
        <dataset name>,
        <dataset type>,
        <data item row size>,
        <data item column size>,
        <data set definition error code>

<data set name>:: = '<name>' | <variable>
<data set type>:: = '<type specification>' | <variable>
<type specification>:: = INT | REAL | DOUB
                        IVEC | RVEC | DVEC
                        IMAT | RMAT | DMAT

<data item row size>:: = <size> | <empty> | VAR
<data item column size>:: = <size> | <empty> | VAR
<size>:: = <unsigned integer> | <variable>
<data set definition error code>:: = <variable>
<relation definition statement>:: =
        <reserved procedure DRLDFN>
        (<relation definition parameter part>)

<relation definition parameter part>:: =
        <database name>,
        <relation name>,
        <attribute name array>,

```



```

        <attribute type array>,
        <attribute row size array>,
        <attribute column size array>,
        <attribute key specification array>,
        <relation definition error code>

<relation name>:: = '<name>'|<variable>
<attribute name array>:: = '<name array>'|<variable>
<name array>:: = <name>|<name><name array>
<attribute type array>:: = '<type specification array>'|<variable>
<type specification array>:: = <type specification>
                                |<type specification><type specification array>
<attribute row size array>:: = <variable>
<attribute column size array>:: = <variable>
<attribute key specification array>:: =
                                <key specification array>|<variable>
<key specification array>:: = KEY|KEY<key specification>|<empty>
<relation definition error code>:: = <variable>
<data definition termination statement>:: =
                                <reserved procedure DSEND>
                                (<data definition termination parameter part>)
<database definition parameter part>:: = <empty>
<data redefinition statement>:: =
                                <reserved procedure DSRDFN>
                                (<data redefinition parameter part>)
<data redefinition parameter part>:: =

```



```

        <data set definition parameter part>
        |<relation definition parameter part>
        |<numerical data definition parameter part>
<matrix data definition statement>:: =
        <reserved procedure DSMATX>
        (<matrix data definition parameters part>)
<matrix data definition part>:: =
        <database name>
        <matrix identification>
        <matrix characteristics>
        <matrix definition error code>
<matrix identification>:: = '<name>'|<variable>
<matrix characteristics>:: =
        'SQUARE', <S. Details>|
        'BANDED', <B. Details>|
        'HYPERMATRIX', <H. Details>|
        'SKYLINE', <K. Details>|
        'SPARSE', <P. Details>
<S. Details>:: =
        <matrix storage type>
        <matrix ordering>,
        <matrix size>,
        <empty>, <empty>,
        <empty>, <empty>,
        <empty>, <empty>,
<matrix storage type>:: = '<matrix storage string>'|<variable>

```



```

<matrix storage string>:: = UPPER|LOWER|FULL
<matrix ordering>:: = '<matrix ordering string>'<variable>
<matrix ordering string>:: = ROW|COLUMN
<matrix size>:: =      <row size>,<column size>|VAR,<column size>|
                        row size>,VAR|VAR,VAR
B. Details>:: =      <matrix storage type>,
                        <matrix ordering>,
                        <matrix size>,
                        <matrix band size>,
<matrix band size>:: = <number of upper codiagonals>,
                        <number of lower codiagonals>
<number of upper codiagonals>:: = <unsigned integer>
<number of lower codiagonals>:: = <unsigned integer>
<H. Details>:: =      <matrix storage type>
                        <matrix ordering>,
                        <matrix size>,
                        <submatrix size>
<submatrix size>:: = row size>|<variable>,<column size>|<variable>
<K. Details>:: =      <empty>,
                        <empty>,
                        <matrix size>,
                        <skyline definition>
<skyline definition>:: = <array of skyline height>
<P. Details>:: =      <empty>,
                        <empty>,

```



<matrix size>,  
<empty>,<empty>  
<empty>,<empty>

- 
- Note: 1) Vertical bar | denotes options for choosing items to the left of the bar or to the right of the bar
- 2) [ ] indicate items within it are optional
- 3)  $\langle x \rangle :: = \langle y \rangle \mid \langle x \rangle \langle z \rangle$  denotes a recursive statement. x is used repeatedly
- 4) ' ' indicates items within it taken as data



**APPENDIX III**  
**BNF DESCRIPTION OF THE PROPOSED**  
**DATA MANIPULATION LANGUAGE**



```
<Data manipulation statement>:: =  
    <database open statement>  
    <database close statement>  
    <data retrieval statement>  
    <data append statement>  
    <data modify statement>  
    <data delete statement>  
    <data copy statement>  
    <matrix retrieval statement>  
    <matrix append statement>  
    <matrix modify statement>  
    <matrix delete statement>  
    <matrix copy statement>  
<database open statement>:: =  
    <reserved procedure DBOPEN>  
    (<database open parameter part>  
<database open parameter part>:: =  
    <database name>,  
    <database hierarchy>  
    <database open error code>  
<database open error code>:: = <variable>  
<database close statement>:: =  
    <reserved procedure DBCLOS>  
    (<database close statement>)  
<database close statement>:: =
```



```

    <database name>
    <database hierarchy>
    <database close error code>
<data retrieval statement>:: =
    <reserved procedure DSGET>
    (<data retrieval parameter part>)
<data retrieval parameter part>:: =
    <database name>,
    <data set name>|<relation name>,
    <identification number>|<empty>,
    <user buffer>,
    <data manipulation error code>
<identification number>:: = <tuple number>|<row number>
<tuple number>:: = <unsigned integer>|<variable>
<row number>:: = <unsigned integer>|<variable>
<user buffer>:: = <variable>
<data manipulation error code>:: = <variable>
<data append statement>:: =
    <reserved procedure DSPUT>
    (<data append parameter part>)
<data append parameter part>:: =
    <data retrieval parameter part>
<data modify statement>:: =
    <reserved procedure DSMOD>
    (<data modify parameter part>)

```



<data modify parameter part>:: =

    <data retrieval parameter part>

<data delete statement>:: =

    <database name> ,

    <data set name>|<relation name> ,

    <identification number>|<empty> ,

    <data manipulation error code>

<data copy statement>:: =

    <reserved procedure DSCOPY>

    (<data copy parameter part>)

<data copy parameter part>:: =

    <database name-copy from> ,

    <data set or relation name-copy from>

    <database name-copy to>

    <data set or relation name-copy to>

    <data manipulation error code>

<database name-copy from>:: = <database name>

<data set or relation name-copy from>:: = <data set name>|<relation name>

<database name-copy to >:: = <database name>

<data set or relation name - copy to>:: = <data set name>|<relation name>

<data manipulation error code>:: = <variable>

<matrix retrieval statement>:: =



```

    <reserved procedure MTGET>
    (<matrix retrieval parameters part>)
<matrix retrieval parameter part>:: = <database name>,
    <matrix identification>,
    <row number>,[<column number>],
    [<column number>,<row number>],
    [<empty>,<empty>],
    <user buffer>,
    <matrix manipulation error code>
<matrix manipulation error code>:: = <variable>
<matrix append statement>:: =
    <reserved procedure MTPUT>
    (<matrix append parameter part>)
<matrix append parameter part>:: =
    <matrix retrieval parameter part>
<matrix modify statement>:: =
    <reserved procedure MTMOD>
    (<matrix modify parameter part>)
<matrix modify parameter part>:: =
    <matrix retrieval parameter part>
<matrix delete statement>:: =
    <reserved procedure MTDEL>
    (<matrix delete parameter part>)
<matrix delete parameter part>:: =

```



```
<database name>
<matrix identification>
<row number>,[<column number>],
|<column number>,[<row number>],
|<empty>,<empty>,
<matrix manipulation error code>

<matrix copy statement>:: =
    <reserved procedure MTCOPY>
    (<matrix copy parameter part>)

<matrix copy parameter part>:: =
    <database name-copy from>,
    <matrix identification-copy from>,
    <database name-copy to>,
    <matrix identification-copy to>,
    <matrix manipulation error code>

<matrix identification-copy from>:: = <matrix identification>
<matrix identification-copy to>:: = <matrix identification>
```



## REFERENCES

- Allan III, J.J. (1972), "Foundations of the Many Manifestations of Computer Augmented Design," Computer-Aided Design, Proceedings of International Federation of Information Processing, pp. 27-58.
- Afimiwala, K.A. and Mayne, R.W. (1979), "Interactive Computer Methods for Design Optimization," Computer-Aided design, Vol. 11, No. 4, pp. 201-208.
- Arora, J.S. and Govil, A.K. (1977), "An Efficient Method for Optimal Structural Design by Substructuring," Computers and Structures, Vol. 7, pp. 507-515.
- Arora, J.S., Ryu, Y.S. and Wu, C.C. (1984a), "A User's Manual for the Computer Program DOCS: Level 3.0," Optimal Design Laboratory, College of Engineering, The University of Iowa.
- Arora, J.S., Thanedar, P.B., and Tseng, C.H. (1984b), "User's Manual for Program IDESIGN Version 3.1," Optimal Design Laboratory, College of Engineering, The University of Iowa.
- Balling, R.J., Pister, R.S., Polak, E. (1983), "DELIGHT-STRUCT: An Optimization-Based Computer-Aided Design Environment for Structural Engineering," Computer Methods in Applied Mechanics and Engineering, 38, pp. 237-251.
- Bell, Jean (1982), "Data Modelling of Scientific Simulation Programs," Int. Conf. On Management of data, ACM-SIGMOD, pp. 79-86.
- Bennett, J.A. and Nelson, M.F. (1979), "An Optimization Capability for Automotive Structures," Society of Automotive Engineers Transactions, Vol. 88, pp. 3236-3243.
- Blackburn, C.L., Storaasli, O.O. and Fulton, R.E. (1982), "The Role and Application of Database Management in Integrate Computer-Aided Design," Journal of American Institute of Aeronautics and Astronautics, pp. 603-613.
- Browne, J.C. (1976), "Data Definition, Structures, and Management in Scientific Computing," Proceedings of Institute for Computer Application in Science and Engineering, Conference on Scientific Computing, pp. 25-56.



- Bryant, J.C. (1978), "A Data Management System for Weight Control and Design-to-Cost", NASA Conference Publication 2055, pp. 65-84.
- Buchmann, A.P. and Dale, A.G. (1979), "Evaluation Criteria for Logical Database Design Methodologies," Computer-Aided Design, pp. 121-126.
- Comfort, D.L. and Erickson, W.J. (1978), "RIM-A Prototype For A Relational Information Management System," NASA Conference Publications 2055, pp. 183-196.
- Cook, R.D. (1981), "Concepts and Application of Finite Element Analysis", John Wiley and Sons, New York.
- Czekalinski, L. and Zgorzelski, M. (1982), "Design Database Organization and Access Problems in Large Scale Machine Manufacturing Industry," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 297-308.
- Daini, O.A. (1982), "Numerical Database Management System: A Model," International Conference of Management of Data, Association for Computing Machinery Special Interest Group Management of Data.
- Darby-Downman, K. and Mitra, G. (1983), "Matrix Storage Schemes In Linear Programming," Special Interest Group Management of Application Programs Bulletin Association for Computing Machinery, No.32, pp. 24-38.
- Date, C.J. (1977), An Introduction To Database Systems, Addison-Wesley, Reading, Mass.
- Derwa, G.T. (1978), "Advanced Program Weight Control System," NASA Conference Publication 2055, pp. 55-64.
- Eastman, C.M. (1978), "The Representation of Design Problems and Maintenance of Their Structure," Artificial Intelligence and Pattern Recognition in Computer-Aided Design, Proceedings of International Federation of Information Processing, pp. 335-366.
- Eastman, C.M. and Fenves, S.J. (1978), "Design Representation and Consistency Maintenance Needs in Engineering Databases," NASA Conference Publication 2055, pp. 1-18.
- Eastman, C.M. and Henrion, M. (1980), "The Glide Language for CAD," J. Of the Technical Councils of American Society of Civil Engineers, Vol. 106, No. TC1, pp. 171-184.
- Eberlein, W. and Wedekind, H. (1982), "A Methodology for Embedding Design Databases into Integrated Engineering Systems," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 3-37.



- Elliott, L., Kunii, H.S., and Browne J.C. (1978), "A Data Management System For Engineering and Scientific Computing," NASA Conference Publications 2055, pp. 197-222.
- Emkin, L.Z. (1978), "ICES Cocepts-A Modern System Approach," Computing In Civil Engineering pp. 89-107.
- Encarnação, J. and Schlechtendahl, E.G. (1983), Computer-Aided Design, Springer-Verlag, Berlin.
- Felippa, C.A. (1979), "Database Management In Scientific Computing-I General Description," Computers and Structures, Vol. 10, pp. 53-61.
- Felippa, C.A. (1980), "Database Management In Scientific Computing-II, Data structures and Program architecture," Computers and Structures, Vol. 12, pp. 131-145.
- Felippa, C.A. (1982), "Fortran-77 Simulation of Word-addressable Files," Advances in Engineering software, Vol. 4, No. 4, pp. 156-162.
- Fischer, W.E. (1979), "PHIDAS -a Database Management System for CAD/CAM Software," Computer-Aided Design, Vol 11, No. 3, pp. 146-150.
- Fishwick, P.A. and Blackburn, C.L. (1982), "The Integration Engineering Programs using a Relational Database Scheme," Computers In Engineering, Internation Computer Engineering Conference, pp. 173-181.
- Fleury, C., Ramanathan, R.K., Salana, M. and Schmit, Jr., L.A. (1981), "ACCESS Computer program for the Synthesis of Large Structural Systems," Proceedings of the International Symposium on Optimum Structural Design, University of Arizona, Tucson.
- Foissey, J. and Valette, F.R. (1982), "A Computer Aided Design Data Model: FLOREAL," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 315-330.
- Fulton, R. E. and Voigt, S.J. (1976), "Computer-Aided Design and Computer Science Technology," Third Institute for Computer Application in Science and Engineering Conference on Scientific Computing, pp. 57-82.
- Galletti, C.U. and Giannotti, E.I. (1981), "Interactive Computer System Functional Design Of Mechanisms," Computer-Aided Design, Vol. 13, No. 3, pp. 159-163.
- Giles, G.L. and Haftka, R.T. (1978), "SPAR Data Handling Utilities," NASA Technical Memorandum 78701.



- Grabowski, H., Eigner, M. and Rausch, W. (1978), "CAD Data-Structures For Minicomputers," Third International Conference on Computers and Engineering, pp. 530-548.
- Grabowski, H. and Eigner, M. (1979), "Semantic Datamodel Requirements and Realization with a Relational Data Structure," Computer-Aided Design, Vol. 11, No. 3, pp. 158-167.
- Grabowski, H. and Eigner, M. (1982), "A Data Model for a Design Database," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 117-144.
- Haskin, R.L. and Lorie, R.A. (1982), "On extending the Functions Of a Relational Database System," International Conference on Management of Data Association of Computing Machinery, pp. 207-212.
- Haug, E.J. and Arora, J.S. (1979), "Applied Optimal Design," Wiley Interscience, John Wiley and Sons, Inc., New York.
- Heerema, F.J. and van Hedel, H. (1983), "An Engineering Data managemet System for Computer-Aided Design", Advances in Engineering Software, Vol. 5, No. 2, pp. 67-75.
- Jefferson, D.K. and Thomson, B.M. (1978), "Engineering Data Management: Experience and Projections," NASA Conference publication 2055, pp. 223-242.
- Jenne, R.L. and Joseph, D.H. (1978), "Management of Atmospheric Data", NASA Conference Publication 2055, pp. 129-140.
- Johnson, H.R., Comfort, D.L. and Shull, D.D. (1980), "An Engineering Data Management System for IPAD," IPAD: Integrated Programs for Aerospace-vehicle Design, NASA Conference Publication 2143, pp. 145-178.
- Jumarie, G. (1982), "A Decentralized Database via Micro-computers a Preliminary Study," Computers in Engineering, Int. comp. Engg. Confer. ASME, pp. 183-187.
- Kamel, H.A., McCabe, M.W. and Spector, W.W. (1979), GIFTS5 System Manual, University of Arizona, Tucson.
- Koriba, M. (1983), "Database Systems: Their Applications to CAD Software Design," Computer-Aided Design, Vol. 15, No. 5, pp. 277-288.
- Kunni, T.I. and Kunni, H.S. (1979), "Architecture of a Virtual Graphic Database System For Interactive CAD", Computer-Aided Design, Vol. 11, No. 3, pp. 132-135.



- Kutay, A.R. and Eastman, C.M. (1983), "Transaction Management in Engineering Databases," Engineering Design Applications, Proceedings of Annual Meeting, Database Week, Association for Computing Machinery Special Interest Group Management of Data, pp. 73-80.
- Lafue, G. (1978), "Design Database and Data Base Design," Third Int. Conf. On computers in Engg. and Building Design CAD78, Brighton Metropole, Sussex, U.K., 14-16.
- Lafue, G.M.E. (1979), "Integrating Language Database for CAD Applications," Computer-Aided Design, Vol. 11, No. 3, pp. 127-129.
- Leinemann, K. and Schlechtendahl, E.G. (1976), "The Regent System for CAD," CAD Systems, Proceedings of International Federation of Information Processing, pp. 143-168.
- Lillehagen, F.M., and Dokken, T. (1982), "Towards a Methodology for Constructing Product Modelling Databases in CAD," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 59-88.
- Lopatka, R.S. and Johnson, T.G. (1978), "CAD/CAM Data Management Needs, Requirements and Options," NASA Conference Publications 2055, pp. 25-40.
- Lopez, L.A. (1974), "FILES: Automated Engineering Data Management System," Computers in Civil Engineering, Electronic Computation, pp. 47-71.
- Lopez, L.A., Dodds, R.H., Rehak, D.R. and Urzua, J.L. (1978), "Application of Data Management to Structures," Computing in Civil Engineering, pp. 477-498.
- Managaki, M. (1982), "Multi-layered Database Architecture for CAD CAM Systems," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 281-290.
- Martin, J. (1977), "Computer Database Organization", Prentice-Hall, Inc., Englewood Cliff, N.J.
- Massena, W.A. (1978), "SDMS - A Scientific Data Management System," NASA Conference Publication 2055, pp. 143-154.
- Nye, W. (1981), "DELIGHT- Design Language with Interactive Graphics and a Happier Tomorrow," Electronics Research Laboratory, University of California, Berkeley, CA.
- Pahl, P.J. (1981), "Data Management in Finite Element Analysis," Nonlinear Finite Element Analysis in Structural Mechanics, Wunderlich, W., Stein, E. and Bathe, K.J., Eds., Springer-Verlag, Berlin, pp. 714-716.



- Pooch, U.W. and Neider, A. (1973), "A Survey Of Indexing Techniques For Sparse Matrices," Computing Surveys, Vol. 5, No. 2, pp. 109-133.
- Rajan, S.D. and Bhatti, M.A. (1983), "Data Management in FEM-Based Optimization Software," Computers and Structures, Vol. 16, No. 1-4, pp. 317-325.
- RIM User's Guide (1982), Boeing Commercial Airplane Company, P.O. Box 3707, Seattle, Washington, 98124.
- Ronald, D.P. (1978), "XIO-A Fortran Direct Access Data Management System," NASA Conference Publication 2055, pp. 155-162.
- Roos, D. (1966), "ICES System Design", The M.I.T. Press, Massachusetts.
- Roussopoulos, N. (1979), "Tool for Designing Conceptual Schemata of Databases," Computer-Aided Design Vol. 11, No. 2, pp. 119-120.
- Ryu, Y.S., Haririan, M., Wu, C.C., and Arora, J.S. (1985), "Structural Design Sensitivity Analysis of Nonlinear Response," Computers and Structures, Vol. 21, No. 1/2, pp. 245-255.
- Schrem, E. (1978), "Functional Software Design and its Graphical Representation," Computers and Structures Vol. 8, pp. 491-502.
- Shenoy, R.S. and Patnaik, L.M. (1983), "Data Definition and Manipulation Languages for a CAD Database," Computer-Aided Design, Vol. 15, No. 3, pp. 131-134.
- Shyy, Y-K. (1985), "A Database Management System for Engineering Applications," M.S. Thesis, Optimal Design Laboratory, College of Engineering, The University of Iowa.
- Sobieszczanski-Sobieski, Jaroslaw (1980), "From a Black-Box to a Programming System: Remarks on Implementation and Application of Optimization Methods," Proceedings of a NATO Advanced Study Institute Session on Structural Optimization, Sart-Tilman, Belgium.
- Somekh, E. and Kirsch, U. (1979), "Interactive Optimal Design of Truss Structures," Computer-Aided Design pp. 253-258.
- Southall, J.W. (1980), "Requirements for Company-Wide Management of Engineering Information," IPAD: Integrated Programs for Aerospace-vehicle Design, NASA Conference Publication 2143, pp. 59-74.
- Sreekanta Murthy, T. and Arora, J.S. (1983), "A Simple Database Management Program (DATHAN)," Technical Report, Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA.



- Sreekanta Murthy, T. and Arora, J.S. (1983), "Database Management Concepts In Design Optimization," Technical Report, Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA.
- Sreekanta Murthy, T., Reddy, C.P. and Arora, J.S. (1983), "User's Manual For Engineering Database Management System EDMS," Technical Report, Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA.
- Sreekanta Murthy, T., Reddy, C.P.D. and Arora, J.S. (1984), "Database Management Concepts in Engineering Design Optimization," Proceedings of AIAA/ASME/ASCE/AHS 25th Structures, Structural Dynamics and Material Conference.
- Sreekanta Murthy, T. and Arora, J.S. (1985), "A Survey of Database Management in Engineering," Journal of Advances in Engineering Software, Vol. 7, No. 3, pp. 126-132.
- Sreekanta Murthy, T. and Arora, J.S. (1985), "A Methodology to Design Databases for Finite Element Analysis and Structural Design Optimization Applications," Proceedings of AIAA/ASME/ASCE/AHS 26th Structures, Structural Dynamics and Material Conference, submitted for publication in the Journal of Computers in Engineering.
- Sreekanta Murthy, T., Shyy, Y-K. and Arora, J.S. (1986), "MIDAS: Management of Information for Design and Analysis of Systems," Journal of Advances in Engineering Software, (to appear).
- Sreekanta Murthy, T. and Arora, J.S. (1986), "Database Management Concepts in Computer-Aided Design Optimization," Journal of Advances in Engineering Software, (to appear).
- Thanedar, P.B., Park, G.J., and Arora, J.S. (1985), "Performance of Two Superlinearly Convergent RQP Optimization Algorithm on Some Structural Design Programs," Technical Report ODL-85.12, Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA.
- Ulfshy, S., Steiner, S. and Oian, J. (1979), "TORNADO: A DBMS for CAD/CAM Systems," Computer-Aided Design, pp. 193-197.
- Valle, G. (1976), "Relational Data Handling Techniques in Computer-Aided Design Process," CAD Systems, Proceedings of International Federation of Information Processing, pp. 309-326.
- Vetter, M. and Maddison, R.N (1981), "Database Design Methodology", Prentice/Hall International.
- Whetstone, W.D. (1977), SPAR Structural Analysis System Reference Manual, System Level II, Vol. I, NASA CR-145098-1.



END

12-86

DTIC